RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Übungsblatt 1
Emmes—Langer—Reidl—Rossmanith 11.04.2011

Übung zur Vorlesung Datenstrukturen und Algorithmen

Aufgabe T1

Berechnen Sie eine möglichst einfache Funktion $f(n)$ mit $\left(\frac{n}{2} + \log(n^2)\right)^3 = f(n) + O(n^2)$.

Lösungsvorschlag

$$\left(\frac{n}{2} + \log(n^2)\right)^3 = \frac{n^3}{8} + \frac{3n^2}{4}\log(n^2) + O(n^2) = \frac{n^3}{8} + \frac{3n^2}{2}\log(n) + O(n^2)$$

Aufgabe T2

Schätzen Sie $\log(n^2 + 3n)$ bis auf einen additiven Term von $O(1/n)$ ab.

Lösungsvorschlag

Mit der wichtigen Taylorreihe $\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \cdots = \sum_{n=1}^{\infty}(-1)^{k+1}\frac{x^k}{k}$ für $-1 < x \leq 1$ gilt:

$$\log\Big(n^2 + 3n\Big) = \log\Big(n^2\Big(1 + \frac{3}{n}\Big)\Big) = 2\log n + \log\Big(1 + \frac{3}{n}\Big) = 2\log n + O\Big(\frac{1}{n}\Big)$$

Aufgabe T3

Beweisen Sie, daß für alle $f\colon \mathbf{R} \to \mathbf{R}$, so daß $|f(n)|$ monoton steigt, gilt: $f(O(1)) = O(1)$. Gilt diese Aussage auch für beliebige f?

Lösungsvorschlag

Sei f wie oben und $g = O(1)$ beliebig. Dann existieren nach Definition Konstanten $c, n_0 \in \mathbf{N}$, so daß $|g(n)| \leq |c|$ für alle $n \geq n_0$. Setze $d = f(|c|)$ und betrachte $n \geq n_0$. Da $|f(n)|$ monoton steigend ist und $|g(n)| \leq |c|$, ist auch $|f(n)| = |f(|g(n)|)| \leq |f(|c|)| = |d|$. Für beliebige f gilt dies jedoch nicht: Für $g\colon x \mapsto 1/x$ gilt $g(n) = O(1)$, da $g(n) \leq 1$ für alle $n \geq 1$. Wenn obige Gleichung gölte, dann wäre also auch $f(g(n)) = O(1)$. Mit $f = g$ sieht man aber leicht, daß $f(g(n)) = n$ ein Gegenbeispiel darstellt.

Aufgabe T4

Zeigen Sie, daß $2^{\sqrt{n^2+5}} = 2^n\,(1 + O(1/n))$ ist.

Lösungsvorschlag
Mit Hilfe der Taylorentwicklungen sieht man für $z \to 0$ leicht ein: $\sqrt{(1+z)} = 1 + O(z)$ und $e^z = 1 + O(z)$. Damit erhält man direkt:

$$2^{\sqrt{n^2+5}} = 2^{\sqrt{n^2(1+5/n^2)}} = 2^{n\sqrt{1+5/n^2}} = 2^{n(1+O(1/n^2))} = 2^{n+O(1/n)} = 2^n(1+O(1/n))$$

Aufgabe H1 (5 Punkte)
Beweisen Sie formal, daß gilt:

$$\frac{1}{O(n)} = \Omega\Big(\frac{1}{n}\Big)$$

Lösungsvorschlag
Betrachten wir eine Funktion $f(n) = O(n)$. Wir müssen nun zeigen, daß $1/f(n) = \Omega(1/n)$, falls $1/f(n)$ existiert (also falls immer $f(n) \neq 0$).
Wegen $f(n) = O(n)$ muß es $N \in \mathbf{N}$ und $c > 0$ geben, so daß $|f(n)| \leq c|n|$ für alle $n \geq N$ gilt. Dann ist aber auch $1/|f(n)| \geq 1/(c|n|)$ für ebenfalls alle $n \geq N$. Wir haben also ein $N' \in \mathbf{N}$ (nämlich N) und ein $c' > 0$ (nämlich $1/c$) gefunden, so daß gilt:
$|1/f(n)| \geq c' \cdot |1/n|$ für alle $n \geq N'$.
Damit ist also nach Definition $1/f(n) = \Omega(1/n)$.

Aufgabe H2 (10 Punkte)
Sei $H_n = \sum_{k=1}^{n} \frac{1}{k}$. Beweisen Sie, welche der folgenden Aussagen jeweils gilt:

$$\begin{aligned} n^{\log n} &= O((\log n)^n) & \qquad n^{\log n} &= \Omega((\log n)^n) \\ (\sqrt{n})^{n^2} &= O((n^2)^n) & \qquad (\sqrt{n})^{n^2} &= \Omega((n^2)^n) \\ H_n &= O(\ln n) & \qquad H_n &= \Omega(\ln n) \end{aligned}$$

Lösungsvorschlag
Es ist $n^{\log n} = 2^{(\log n)^2}$ und $(\log n)^n = 2^{n \log\log n}$. Für $n \geq 32$ ist $n > (\log n)^2$ (man vergleiche $\sqrt{n}$ und $\log n$) und daher $n = \Omega((\log n)^2)$. Somit gilt erst recht $n \log\log n = \Omega((\log n)^2)$, da $\log\log n \geq 1$ für $n \geq 4$ ist. Umgekehrt ist für jede Konstante $c \in \mathbf{R}$ und genügend große n aber $2^{cn} = n' > c \log n' \log n' = c \log 2^{cn} \log 2^{cn} = c(cn)^2$, so daß keine c, n_0 mit $n \log\log n \leq c(\log n)^2$ für $n > n_0$ existieren können. Daher ist $n^{\log n} = O((\log n)^n)$ korrekt und $n^{\log n} = \Omega((\log n)^n)$ nicht.

Auf ähnliche Weise erhält man $(\sqrt{n})^{n^2} = n^{\frac{n^2}{2}}$ und $(n^2)^n = n^{2n}$. Weil $\frac{n^2}{2} = \Omega(n)$ gilt ($n^2/2 > n$ für alle $n \geq 3$), ist $(\sqrt{n})^{n^2} = \Omega((n^2)^n)$ korrekt. Offenbar ist für jedes $c \in \mathbf{R}$ und $n > 1$ jedoch $c^2n^2 = n'^2 > cn' = c^2n$. Daher gilt *nicht* $n^2/2 = O(n)$, und $(\sqrt{n})^{n^2} = O((n^2)^n)$ erst recht nicht.
Die letzten Aussagen sind beide wahr: es gilt nämlich $H_n = \Theta(\ln n)$. Betrachten wir dazu die folgende Darstellung des Logarithmus:

$$\ln n = \int_1^n \frac{1}{x} dx$$

Für diese können wir leicht eine Abschätzung durch geeignete Ober- und Untersummen durchführen, indem wir das Integral der Funktion $\frac{1}{x}$ durch Streifen der Breite eins annähern (man überzeuge sich von der Richtigkeit dieser Summen durch eine Skizze):

$$\begin{aligned} \sum_{k=1}^{n-1} \frac{1}{k+1} \leq & \int_1^n \frac{1}{x} dx & \leq \sum_{k=1}^{n-1} \frac{1}{k} \\ \Leftrightarrow \sum_{k=1}^{n} \frac{1}{k} - 1 - \frac{1}{n+1} \leq & \int_1^n \frac{1}{x} dx & \leq \sum_{k=1}^{n} \frac{1}{k} - \frac{1}{n} \\ \Leftrightarrow H_n - (1 + \frac{1}{n+1}) \leq & \ln n & \leq H_n - \frac{1}{n} \end{aligned}$$

Für $n > 0$ sind die Terme $1 + \frac{1}{n+1}$ und $\frac{1}{n}$ durch Konstanten beschränkt, folglich ist $H_n = \Theta(\ln n)$.

Aufgabe H3 (5 Punkte)
Betrachten Sie folgenden Algorithmus:

```
for i = 1 to n do
  for j = i to n do
    for k = i to j do
      print (i, j, k)
```

Finden Sie ein $f\colon \mathbf{N} \to \mathbf{N}$, so daß die Anzahl ausgegebener Zeilen $\Theta(f(n))$ ist.

Lösungsvorschlag
Wir wagen die Vermutung, daß $f(n) = n^3$ ist. Eine grobe Abschätzung der obigen Laufzeit ergibt, daß jede Schleife höchstens n-mal durchlaufen wird, es werden demnach $O(n^3)$ Zeilen ausgegeben. Es bleibt zu zeigen, daß es auch mindestens $\Omega(n^3)$ Zeilen sein müssen. Betrachten wir den Fall von $i \in \{1 \ldots \frac{n}{3}\}$ und $j \in \{\frac{2n}{3} \ldots n\}$. Es ist leicht zu sehen, daß der obige Algorithmus diese Konfiguration für angenehm große n immer durchlaufen muss (um verwirrendes Runden zu vermeiden, nehmen wir an, daß n durch drei teilbar sei). Weil k eben genau die ganzen Zahlen von i bis j durchläuft, wird die Anweisung `print(i,j,k)` demnach mindestens

$$\left|\left\{1, \ldots, \frac{n}{3}\right\}\right| \cdot \left|\left\{\frac{2n}{3}, \ldots, n\right\}\right| \cdot \left|\left\{\frac{n}{3}, \ldots, \frac{2n}{3}\right\}\right| = \frac{1}{3}n^3$$

mal ausgegeben. Also ist die Anzahl der ausgegebenen Zeilen $\Theta(n^3)$.

RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Übungsblatt 2
Emmes—Langer—Reidl—Rossmanith 18.04.2011

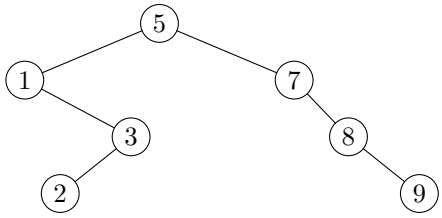
Übung zur Vorlesung Datenstrukturen und Algorithmen

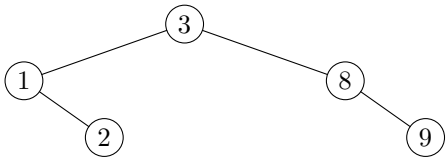
Aufgabe T5

Fügen Sie in einen zu Beginn leeren binären Suchbaum die Elemente 5, 1, 3, 7, 8, 9 und 2 ein (in dieser Reihenfolge). Löschen Sie dann die Elemente 7 und 5 (in dieser Reihenfolge).

Lösungsvorschlag

Nach dem Einfügen und Löschen ergeben sich diese Suchbäume:





Aufgabe T6

Wie verhält sich im allgemeinen die Höhe eines binären Suchbaums, wenn Elemente gesucht, eingefügt oder gelöscht werden?

Lösungsvorschlag

Das Durchsuchen eines binären Suchbaums verändert den Suchbaum nicht. Wenn ein Element eingefügt wird, kann die Höhe höchstens um eins wachsen, nämlich dann, wenn das neu eingefügte Element an ein Blatt auf dem bisher unterstem Level des Suchbaums eingehängt wird.

Beim Löschen eines Elementes kann sich die Höhe höchstens um eins verringern: Wenn ein Blatt gelöscht wird, verringert sich die Höhe des Suchbaums höchstens um eins. Wenn ein innerer Knoten gelöscht wird, tritt an seine Stelle das größte Element im linken Teilbaum des gelöschten Elements. Dadurch ändert sich die Höhe zunächst nicht. Rekursiv wird dann jedoch das so verschobene Element im linken Teilbaum gelöscht. Per Induktion ergibt sich, daß sich die Höhe dieses Teilbaums durch diese Operation dann um höchstens eins verringert, und damit auch die Gesamthöhe des binären Suchbaums.

Aufgabe T7

Gegeben sei eine einfach verkettete Liste mit n Elementen. Entwerfen Sie einen Algorithmus, mit dem sich testen läßt, ob die Liste einen Kreis enthält. Wie ist seine Laufzeit? Ist dieses auch in Zeit $O(n)$ möglich?

Lösungsvorschlag

Eine Möglichkeit sieht so aus: Mit zwei Zeigern l_1, l_2 durchlaufen wir die Liste. In jedem Schritt wird, falls möglich, l_1 um ein Element weiterverschoben, l_2 um zwei Elemente. Wenn dies nicht möglich ist und das Ende der Liste erreicht ist, liegt offenbar kein Zykel vor. Wenn dieses möglich ist, wird verglichen, ob $l_1 = l_2$ ist, also l_1 und l_2 auf das gleiche Element zeigen. Wenn ja, wurde ein Zykel gefunden.

Zur Analyse: Wenn die Liste keinen Zykel enthält, terminiert das Verfahren, sobald das Ende der Liste erreicht wurde. Wenn die Liste jedoch einen Zykel der Länge k enthält, dann ist nach höchstens $2n$ Iterationen die Abbruchbedingung $l_1 = l_2$ erfüllt: Nach höchstens $n - k$ Schritten befinden sich beide Zeiger bereits im Zykel. Nach höchstens k weiteren Schritten hat sich mindestens einmal die Situation ergeben, daß $l_1 = l_2$, denn irgendwann wird l_2 l_1 überholen. D.h., irgendwann wird die Situation entstehen, daß sie höchstens ein Feld weit auseinanderstehen, also $l_2 = l_1 - 1$. Dann gilt im nächsten Schritt aber $l_2 = l_1$.

Aufgabe T8

In der Vorlesung wurde binäre Suche analysiert. Wir müssen nun noch die letzte Lücke schließen und

$$\lfloor \log n \rfloor = \left\lfloor \log \lceil (n-1)/2 \rceil \right\rfloor + 1$$

für $n > 1$ beweisen. Mit $\lfloor$ und $\lceil$-en rechnen zu müssen ist eine typische Schwierigkeit beim Entwurf von Algorithmen.

Lösungsvorschlag

Sei $k \geq 0$, so daß entweder $n = 2k$ oder $n = 2k + 1$. In beiden Fällen ist $\lceil (n-1)/2 \rceil = k$. Dann ist

$$\left\lfloor \log \lceil (n-1)/2 \rceil \right\rfloor + 1 = \left\lfloor \log k + \log 2 \right\rfloor = \left\lfloor \log 2k \right\rfloor$$

Für $n = 2k$ ist die Aussage damit bereits gezeigt. Für ungerade $n = 2k + 1$ sei $q \geq 0$ maximal mit $2^q < n$, also $q = \lfloor \log n \rfloor$. Wegen $2^q \leq 2k$ gilt dann auch

$$q \leq \left\lfloor \log 2k \right\rfloor \leq \left\lfloor \log n \right\rfloor = q.$$

Aufgabe H4 (4 Punkte)

Am Bartresen einer Kneipe in der Pontstraße spricht Sie ein Unbekannter an. Er behauptet: Wenn man in einen zu Beginn leeren binären Suchbaum die gleichen Elemente in unterschiedlichen Reihenfolgen einfügt, dann sind die entstehenden binären Suchbäume danach verschieden. Finden Sie einen Beweis oder ein Gegenbeispiel für diese Behauptung. Nach Möglichkeit sollten sie nicht mehr Platz verwenden, als ein handelsüblicher Bierdeckel bietet.

Lösungsvorschlag

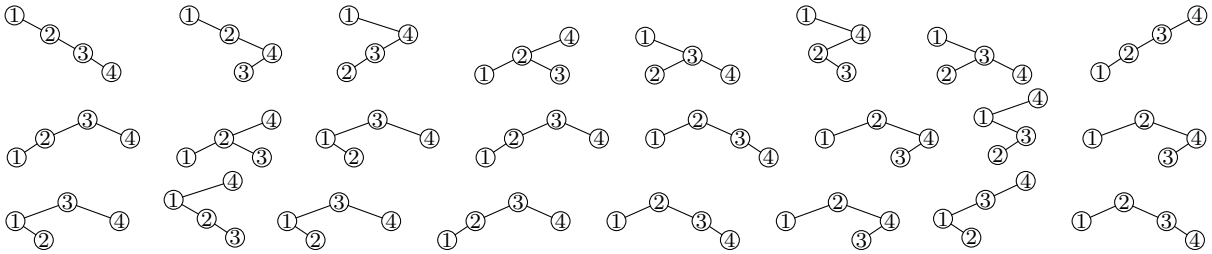
Wenn man die Elemente 1, 5, 8 in den Reihenfolgen 5, 1, 8 bzw. 5, 8, 1 in einen leeren binären Suchbaum einfügt, entsteht jedesmal der gleiche Baum.

Aufgabe H5 (6 Punkte)

Die Zahlen 1, 2, 3 und 4 werden in zufälliger Reihenfolge in einen zu Beginn leeren binären Suchbaum eingefügt. Welches ist die erwartete Höhe des entstehenden Suchbaums? Die erwartete Höhe ist die durchschnittliche Höhe der Suchbäume über alle möglichen Einfügereihenfolgen.

Lösungsvorschlag

Wir können natürlich einfach alle $4! = 24$ Möglichkeiten aufzählen:



Wir sehen, daß 16 Bäume die Höhe drei und acht Bäume die Höhe vier haben. Im Durchschnitt ergibt das genau 10/3.

Etwas weniger Arbeit hat man auf die folgende Weise:

Wenn die 1 zu Beginn eingefügt wird, dann werden die restlichen drei Elemente immer im rechten Teilbaum unterhalb der Wurzel zu finden sein. Die erwartete Höhe über alle Suchbäume, bei denen die 1 zu Beginn eingefügt wird, ergibt sich daher als $1 + h_3$, wobei h_3 die erwartete Höhe des rechten Teilbaums ist, wenn man die verbleibenden drei Elemente in zufälliger Reihenfolge einfügt. Durchprobieren aller sechs möglichen Reihenfolgen 234,243,324,342,423,432 ergibt eine erwartete Höhe von $h_3 = 8/3$.

Wenn die zwei zu Beginn einfügt wird, ist die 1 immer linkes Kind der 2, und es ergeben sich zwei verschiedene Bäume in Abhängigkeit der Reihenfolgen von 3 und 4; in beiden Fällen hat der Suchbaum jedoch Höhe 3.

Die verbleibenden Fälle mit einer 3 oder 4 zu Beginn ergeben sich aus Symmetriegründen analog. Zusammen ergbit sich eine erwartete Höhe von $(11/3 + 3 + 3 + 11/3)/4 = 10/3$.

Aufgabe H6 (10 Punkte)

Entwerfen Sie einen Algorithmus, welcher in Zeit $O(n)$ testet, ob zwei binäre Suchbäume mit jeweils n Elementen dieselben Elemente enthalten.

Lösungsvorschlag

Die wohl einfachste Methode ist es, den Inhalt der Bäume zunächst in sortierte Arrays zu überführen. Der Vergleich der Arrays ist dann mit zwei Zeigern in $O(n)$ durch simples durchlaufen zu bewältigen.

Es bleibt zu zeigen, daß diese Überführung in Zeit $O(n)$ machbar ist. Betrachten wir folgenden Algorithmus, der einen binären Suchbaum rekursiv in ein globales Array `content[]` schreibt und dabei die globable Variable `index` benutzt, um sich die nächste freie Arrayposition zu merken:

```
void flattenTree(TreeNode current) {
    if(current.left ≠ null)
        flattenTree(current.left);
```

```
    content[index] = current.key;
    index++;
    if(current.right ≠ null)
        flattenTree(current.right);
}
```

Rufen wir `flattenTree` auf dem Wurzelknoten auf (welcher vorhanden sei), so wird diese Funktion für jeden Knoten des Baumes genau einmal aufgerufen—womit eine Laufzeit von $O(n)$ gegeben ist.

RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Übungsblatt 3
Emmes—Langer—Reidl—Rossmanith 26.04.2011

Übung zur Vorlesung Datenstrukturen und Algorithmen

Aufgabe T9

Verwenden Sie LuFGTI-Schnipsel, um eine zufällige Permutation der Zahlen $\{1, \ldots, 10\}$ zu erstellen. Dies sollte jeder Teilnehmer und jede Teilnehmerin unabhängig voneinander durchführen. Fügen Sie nun die Zahlen in dieser Reihenfolge in einen leeren Suchbaum ein. Bestimmen Sie die Höhe des entstehenden Suchbaums. Anschließend soll die *durchschnittliche* Höhe aller Bäume in der ganzen Tutorgruppe bestimmt werden.

Lösungsvorschlag:

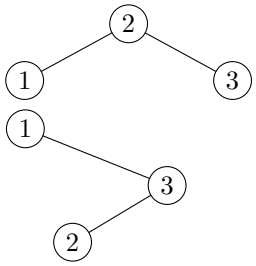
Der genaue Erwartungswert der Höhe ist $\frac{6397}{1134} \approx 5.641$.

Aufgabe T10

Ist es möglich, daß die Zugriffswahrscheinlichkeit des sich in der Wurzel eines optimalen binären Suchbaumes befindlichen Knotens kleiner ist als die aller anderer Schlüssel?

Lösungsvorschlag:

Natürlich. Ein Beispiel ist der optimale Suchbaum für die Elemente 1, 2 und 3 mit Wahrscheinlichkeiten 0.36, 0.28 und 0.36.



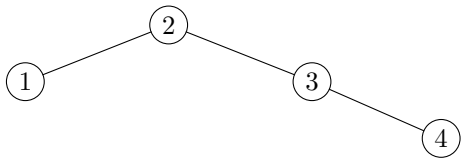
Obacht! Man könnte fälschlicherweise auf die Idee kommen, auch die Wahrscheinlichkeiten 0.49, 0.02 und 0.49 zu wählen, um diesen Baum zu erzeugen. Leider führt dies nicht zum gewünschten Ergebnis...

Aufgabe T11

Konstruieren Sie einen optimalen Suchbaum, der die Schlüssel 1, 2, 3 und 4 enthält. Die jeweiligen Zugriffswahrscheinlichkeiten mögen 1/3, 1/4, 1/4 und 1/6 betragen.

Lösungsvorschlag:

Es ergibt sich folgender optimaler Suchbaum:



$w_{i,j}$	1	2	3	4
1	1/3	7/12	5/6	1
2	0	1/4	1/2	2/3
3	0	0	1/4	5/12
4	0	0	0	1/6

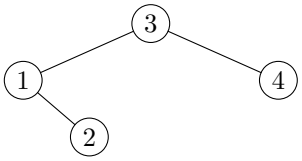
$e_{i,j}$	1	2	3	4
1	1/3 (1)	5/6 (1)	17/12 (2)	23/12 (2)
2	0	1/4 (2)	3/4 (2)	13/12 (3)
3	0	0	1/4 (3)	7/12 (3)
4	0	0	0	1/6 (4)

Aufgabe H7 (10 Punkte)

Konstruieren Sie wieder einen optimalen Suchbaum, der die Schlüssel 1, 2, 3 und 4 enthält. Die jeweiligen Zugriffswahrscheinlichkeiten sollen diesmal aber 3/8, 1/8, 1/4 und 1/4 betragen.

Lösungsvorschlag:

Es ergibt sich folgender optimaler Suchbaum:



$w_{i,j}$	1	2	3	4
1	0.375	0.5	0.75	1.0
2	0.0	0.125	0.375	0.625
3	0.0	0.0	0.25	0.5
4	0.0	0.0	0.0	0.25

$e_{i,j}$	1	2	3	4
1	0.375	0.625	1.25	1.875
2	0.0	0.125	0.5	1.0
3	0.0	0.0	0.25	0.75
4	0.0	0.0	0.0	0.25

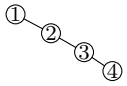
Aufgabe H8 (10 Punkte)

Richtig oder falsch? Wir betrachten einen binären Suchbaum. Die Wurzel sei auf Ebene 1, ihre Kinder auf Ebene 2 et cetera. Wenn für alle i die Zugriffswahrscheinlichkeiten auf Knoten der Ebene i stets kleiner sind als die Zugriffswahrscheinlichkeiten auf Knoten der Ebene $i-1$, dann haben wir bereits einen optimalen Suchbaum.

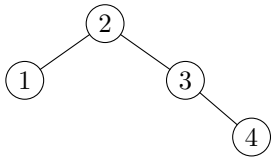
Beweisen Sie diese Behauptung oder finden Sie ein Gegenbeispiel.

Lösungsvorschlag:

Falsch. Wir betrachten einen Suchbaum für die Elemente $1, 2, 3$ und 4 mit den Wahrscheinlichkeiten 0.35, 0.30, 0.20 und 0.15, in dem kein Knoten ein linkes Kind hat.

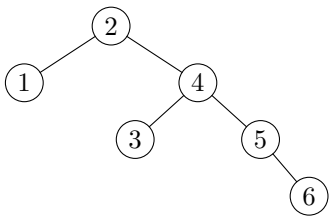


Man sieht leicht, daß dieser nicht optimal ist: Erwartungswert ist $0.35 + 0.6 + 0.6 + 0.6 = 2.15$. Ein optimaler Suchbaum benötigt nur $0.7 + 0.3 + 0.4 + 0.45 = 1.85$ Vergleiche.



Aufgabe H9 (10 Punkte)

Geben Sie Zugriffswahrscheinlichkeiten für die Knoten $1, \ldots, 6$ an, so daß ein möglicher optimaler binärer Suchbaum für diese Schlüssel wie nebenan illustriert aussieht. Beweisen Sie auf möglichst einfache Weise, daß der Suchbaum tatsächlich optimal ist.



Lösungsvorschlag:
Eine einfache Möglichkeit ist es, die folgenden Wahrscheinlichkeiten p_i für das Element i zu wählen: $p_1 = 0$, $p_2 = 1 - 4\epsilon$, $p_3 = p_4 = p_5 = p_6 = \epsilon$ mit $\epsilon = 0.0001$.
Da die 2 deutlich häufiger gesucht wird als die anderen Elemente, muß sie oben stehen: Denn mit $\epsilon = 0.0001$ ist die erwartete Anzahl Vergleiche nur $1 - 4\epsilon + \epsilon(2 + 3 + 3 + 4) \approx 1$. Sobald die 2 mit diesen Wahrscheinlichkeiten nicht in der Wurzel ist, haben wir schon mindestens $2(1 - 4\epsilon)$ Vergleiche. Danach ist es einfach: Die 1 muß in jedem Fall links von er zwei liegen. Für die restlichen vier Elemente benötigen wir auf jeden Fall einen Suchbaum der Höhe drei, der genaue Aufbau ist bei identischen Wahrscheinlichkeiten dann zweitrangig. Der abgebildete Teilbaum ist daher für gleiche Zugriffswahrscheinlichkeiten $p_3 = p_4 = p_5 = p_6 = \epsilon$ optimal.

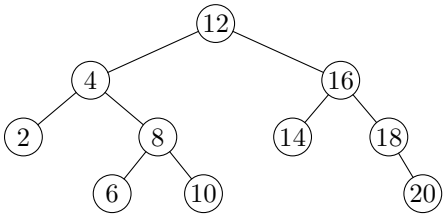
1	2	3	4	5	6	7	8	9	10

RWTH Aachen
Lehrgebiet Theoretische Informatik
Emmes—Langer—Reidl—Rossmanith

SS 2011
Übungsblatt 4
02.05.2011

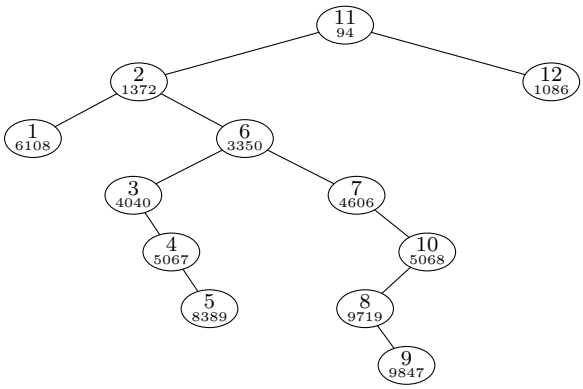
Übung zur Vorlesung Datenstrukturen und Algorithmen

Aufgabe T12 Gegeben ist der folgende AVL-Baum:



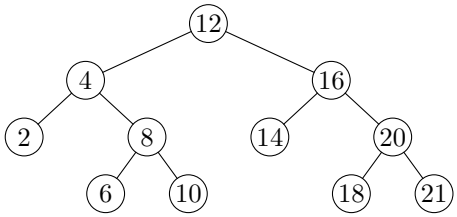
Was passiert, wenn wir erst 21 und dann 17 einfügen, danach erst 12 und dann 2 löschen? Schließlich wird die 12 wieder eingefügt. Wie sieht der Baum am Ende aus?

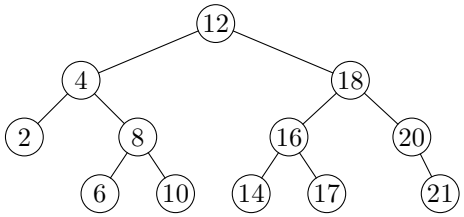
Aufgabe T13

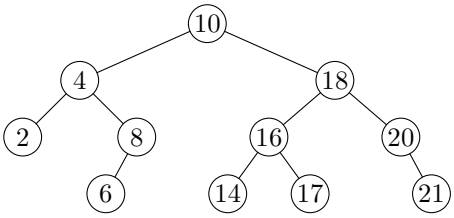


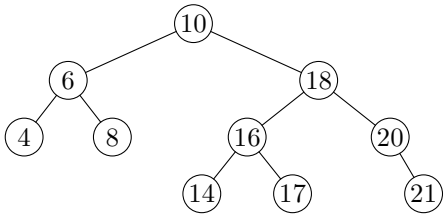
Was passiert, wenn die 11 aus obigen Treap gelöscht und anschließend mit Priorität 3000 wieder eingefügt wird?

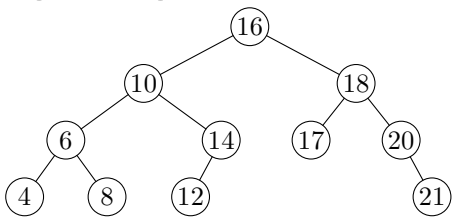
Lösungsvorschlag T12:
Wir sollten uns streng an die in der Vorlesung erörterten Operationen halten. Dann ergeben sich für die obigen fünf Anwendungen die folgenden AVL-Bäume:



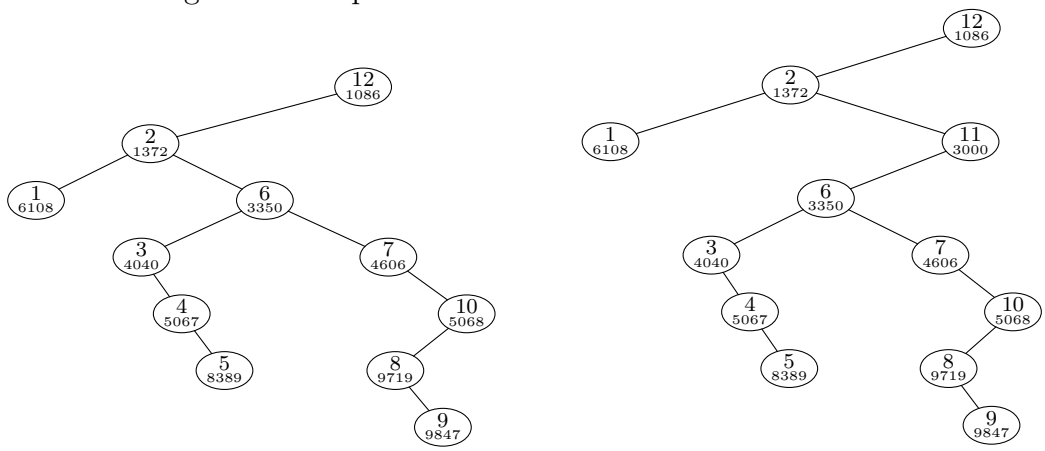








Lösungsvorschlag T13:
Wir sollten uns streng an die in der Vorlesung erörterten Operationen halten. Dann ergeben sich die folgenden Treaps:



Aufgabe T14
Beantworten Sie diese Fragen:

a) Vorgegeben sei ein beliebiger binärer Baum, der die AVL-Eigenschaft erfüllt. Kann man durch die Einfüge- und Löschoperationen für AVL-Bäume einen AVL-Baum mit genau dieser Struktur erzeugen?

b) Können Sie eine Operation *Split* für Treaps angeben, die einen Treap für einen gegebenen Schlüssel s in zwei Treaps T_1, T_2 unterteilt, so daß alle Schlüssel in T_1 kleiner und alle Schlüssel in T_2 größer als s sind?

Lösungsvorschlag:

a) Ja, dies ist immer möglich. Es genügt hierzu, die Elemente des vorgegebenen Baums schichtweise und von der Wurzel aus in einen anfangs leeren AVL-Baum einzufügen. Zu jedem Zeitpunkt ist der so entstehende Baum ein Suchbaum mit der AVL-Eigenschaft.

b) Ja, hoffentlich. Ist der Schlüssel s vorhanden, wird er zunächst gelöscht. Danach wird s mit der Priorität $-\infty$ neu eingefügt und dabei natürlich zur Wurzel durchrotiert. Nun ergeben der linke und rechte Unterbaum die gewünschten Treaps. Die Laufzeit für das Löschen und Einfügen von s ist linear in der Höhe des ursprünglichen Treaps.

Aufgabe T15
Das Array a sei nur mit 0en und 1en gefüllt und repräsentiert auf diese Weise eine Binärzahl. Die Funktion *counterStep* erhöht diese Zahl um eins. Geben Sie die amortisierten Kosten der Funktion *counterStep* an, wenn das Array anfänglich nur mit 0en gefüllt ist.

```
void counterStep (int [ ] a) {
    int i = 0;
    while (a[i] ≡ 1) {
        a[i] = 0;
        i++;
    }
    a[i] = 1;
}
```

Lösungsvorschlag:
Wir verwenden eine Potentialfunktion $\Phi\colon \mathbf{N} \to \mathbf{N}$ definiert vermöge $\Phi(i) = 3b_i$, wobei b_i die Anzahl der 1sen im Array a zum Zeitpunkt i sein soll. Hierbei sei $i = 0$ der Zeitpunkt vor dem ersten Aufruf von *counterStep*, $i = 1$ der Zeitpunkt nach dem ersten Aufruf von *counterStep* usw. Dann ist offenbar wie gewünscht $\Phi(0) = 0$ und $\Phi(i) \geq 0$ für alle i.
Schätzen wir nun zunächst die Anzahl der tatsächlichen Operationen t_i im i-ten Aufruf von *counterStep* ab. Außerhalb der **while**-Schleife werden zwei Operationen getätigt, sowie jedesmal mindestens ein Vergleich für die Bedingung der **while**-Schleife. Die **while**-Schleife selbst wird solange aufgerufen, wie sie von links kommend 1sen vorfindet. Sei k_i die Anzahl der konsekutiven 1sen von links aus gesehen vor dem i-ten Aufruf von *counterStep*. Dann ist also $t_i \leq 3 + 3k_i$.
Da jede der k_i 1sen durch eine 0 ersetzt wird, und danach eine 0 durch eine 1 ersetzt wird, gilt $b_{i-1} - b_i = k_i - 1$. Das Potential ändert sich folglich um

$$\Phi(i) - \Phi(i-1) = 3b_i - 3b_{i-1} = -3k_i + 3.$$

Damit können wir, vorausgesetzt das Array enthält zu Beginn tatsächlich nur 0en, die Gesamtanzahl der Operationen in n Aufrufen von *counterStep* abschätzen als

$$\begin{aligned}
\sum_{i=1}^{n} t_i &\leq \sum_{i=1}^{n} t_i + \underbrace{\overbrace{\Phi(n)}^{\geq 0} - \overbrace{\Phi(0)}^{=0}}_{\text{Teleskopsumme}} \\
&= \sum_{i=1}^{n} t_i + \Phi(i) - \Phi(i-1) \\
&\leq \sum_{i=1}^{n} (3 + 3k_i - 3k_i + 3) = 6n
\end{aligned}$$

Für n Aufrufe ergeben sich also durchschnittlich jeweils weniger als sechs Operationen. Die amortisierten Kosten eines einzelnen Aufrufs von *counterStep* sind folglich $O(1)$.

Aufgabe H10 (10 Punkte)

Einst schickte Frau Mutter, mit eiligen Worten,
klein Timmi zum Markt: „Es gibt neue Waren!
Strukturen für Daten! Verschiedenste Sorten!
Bring mir eine Queue—doch müssen wir sparen.

Drei Groschen für eine, das sollte schon passen."
So lief Timmi fort, den Marktplatz voraus
doch kaum war er dort, konnt' er sich nicht lassen
„Zwei Groschen reichen doch sicherlich aus!"

Von Gier besiegt und mit Eis in der Hand
erschrak Timmi heftig, als er sich besann
„Drei Groschen die Queue" las er dort am Stand—
er zerbrach sich den Kopf und das Eis zerann.

Mit zwei Stacks im Rucksack kehrt er schließlich heim
—wegen reichlicher Ernte bekam er sie beide—
Doch scholt' ihn die Mutter „Das kann doch nicht sein!
Stacks gehen verkehrt, weshalb ich sie meide!"

„Eine Queue, liebe Mutter, bau ich drumherum:
Enqueue-en werd' ich nur in den ersten der beid'
Und will ich dequeue-en, so füll ich sie um.
Amortisiert wird das klappen—in konstanter Zeit."

Hilf klein Timmi! Seine simulierte Queue funktioniert wie folgt:

```
int dequeue() {
  if(right.isEmpty()) {
    while(!left.isEmpty())
      right.push(left.pop());
  }
  return right.pop();
}

void enqueue(int x) {
  left.push(x);
}
```

enqueue legt also nur Elemente auf den linken Stack, *dequeue* dreht dann bei Bedarf den Inhalt des linken Stacks um: Es entfernt sukzessive alle Element und legt sie auf den rechten Stack. Solange der rechte Stack jetzt noch Elemente enthält, nimmt *dequeue* sie schlicht von diesem.

Zeigen Sie, daß eine Operation auf dieser Queue amortisiert nur $O(1)$ Stack-Operationen benötigt. *Hinweis:* Benutzen Sie eine Potentialfunktion $\Phi\colon \mathbf{N} \to \mathbf{N}$, für die folgendes gilt:

1. $\Phi(i)$ ist das Potential *vor* der i-ten Operation, beginnend bei $i = 0$.
2. $\Phi(0) = 0$.
3. $\Phi(i+1) - \Phi(i) = O(1)$, wenn die i-te Operation *enqueue* ist.
4. Wenn die i-te Operation *dequeue* ist, werden nur $O(\Phi(i+1) - \Phi(i)) + O(1)$ Stack-Operationen benötigt.

Lösungsvorschlag:
Nehmen wir an, n Operationen werden auf der Queue ausgeführt. Die i-te Operation benötigt dabei o_i, Stack-Operationen (mit $0 \leq i \leq n-1$). Unterscheiden wir zwischen Enqueue-Operationen e_i und Dequeue-Operationen d_i zum Zeitpunkt i, so können wir diese wie folgt abschätzen:

$$\begin{aligned} e_i &= 1 \\ d_i &\leq \begin{cases} 3|\text{left}_i| + 2 & \text{Wenn der rechte Stack leer ist} \\ 2 & \text{sonst} \end{cases} \end{aligned}$$

Dabei bezeichne $|\text{left}_i|$ die Anzahl der Elemente auf dem linken Stack zum Zeitpunkt i. Wir wollen mit Hilfe der Potentialfunktion $\Phi(i)$ nun e_i und d_i so gegeneinander aufwiegen, daß der amortisierte Aufwand durch eine Konstante beschränkt ist:

$$\begin{aligned} e_i + \Phi(i+1) - \Phi(i) &= 1 + \Phi(i+1) - \Phi(i) \leq c_e \\ d_i + \Phi(i+1) - \Phi(i) &\leq \begin{cases} 3|\text{left}_i| + 2 + \Phi(i+1) - \Phi(i) \leq c_{d2} & \text{Rechst leer} \\ 2 + \Phi(i+1) - \Phi(i) \leq c_{d1} & \text{sonst} \end{cases} \end{aligned}$$

für drei uns noch unbekannte Konstanten c_e, c_{d1} und c_{d2}. Dazu steht uns der Term $3|\text{left}_i|$ noch im Weg, er sollte von der Potentialfunktion geschluckt werden. Der Ansatz $\Phi(i) = 3|\text{left}_i|$ führt uns schnell zum Ziel:

$$e_i + \Phi(i+1) - \Phi(i) = 1 + 3|\text{left}_{i+1}| - 3|\text{left}_i| = 4$$

wobei wir benutzt haben, dass ein `enqueue` ein Element auf den linken Stack legt. Für den Fall eines Dequeues, bei dem Der rechte Stack *nicht* leer ist, gilt

$$d_i + \Phi(i+1) - \Phi(i) = 2 + 3|\text{left}_{i+1}| - 3|\text{left}_i| = 2$$

denn die Größe des linken Stacks ändert sich nicht. Und auch unser Sorgenkind, der Fall in dem der rechte Stack leer ist, benimmt sich hervorragend:

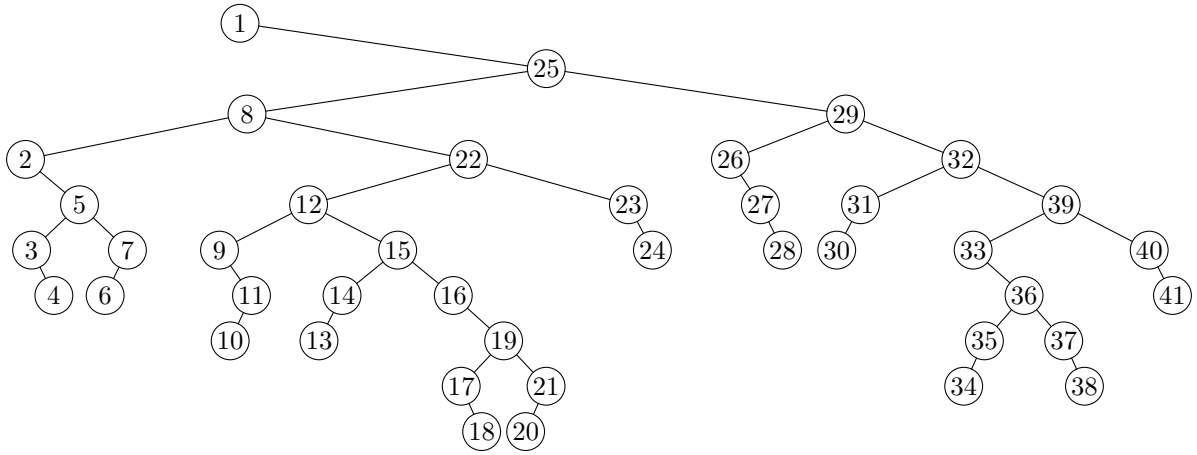
$$\begin{aligned} d_i + \Phi(i+1) - \Phi(i) &\leq 3|\text{left}_i| + 2 + 3|\text{left}_{i+1}| - 3|\text{left}_i| \\ &= 2 + 3|\text{left}_{i+1}| \\ &= 2 \end{aligned}$$

Hier nutzten wir die Tatsache, daß der linke Stack nach der Operation (zum Zeitpunkt $i+1$) leer ist.
So gerüstet können wir nun die Abschätzung von n beliebigen Queue-Operationen $o_i \in \{e_i, d_i\}$ wagen:

$$\begin{aligned} \sum_{i=0}^{n-1} o_i &\leq \left(\sum_{i=0}^{n-1} o_i\right) + \underbrace{\Phi(n-1)}_{\geq 0} - \underbrace{\Phi(0)}_{=0} \\ &= \sum_{i=0}^{n-1} o_i + \underbrace{\sum_{i=0}^{n-1} \Phi(i+1) - \Phi(i)}_{\text{Teleskopsumme}} \\ &= \sum_{i=0}^{n-1} o_i + \Phi(i+1) - \Phi(i) \\ &\leq 4n \end{aligned}$$

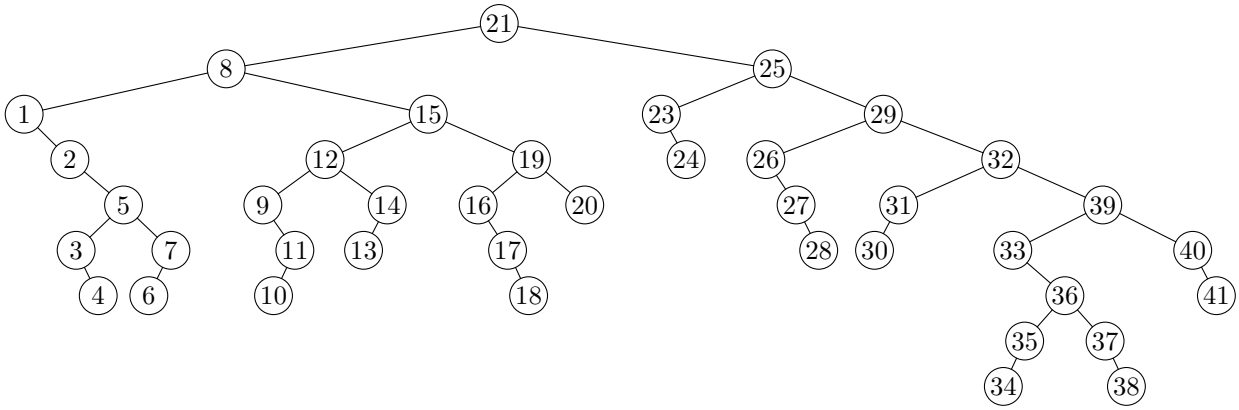
Die Erweiterung durch die Teleskopsumme ist nur noch einmal zur Verdeutlichung angeführt. Damit wissen wir, daß n Operationen auf der Queue maximal $4n$ Stack-Operationen benötigen—im Schnitt (amortisiert) benötigt jede einzelne Queue-Operation also konstant viele Stack-Operationen.

Aufgabe H11 (10 Punkte)
Gegeben sei folgender Splay-Tree:



Es wird der Schlüssel 22 gelöscht. Wie sieht der Splay-Tree anschließend aus?

Lösungsvorschlag:



RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Übungsblatt 5
Emmes—Langer—Reidl—Rossmanith 09.05.2011

Übung zur Vorlesung Datenstrukturen und Algorithmen

Aufgabe T16

Angenommen wir versuchen uns das Leben zu erleichtern, indem wir Splaybäume nicht mit der Zig-Zig-Operation versehen. Stattdessen benutzen wir einfach zwei aufeinanderfolgende Zigs. Finden Sie ein Beispiel, bei welchem dieser Zig-Zig-lose Splaybaum für $O(n)$ Operationen $\Theta(n^2)$ Schritte benötigt. Wie verhält sich der reguläre Splaybaum auf eben diesem Beispiel?

Lösungsvorschlag:

Fügt man die Schlüssel $1, \ldots, n$ in aufsteigender Reihenfolge in einen Splaybaum ein, so ist das Ergebnis ein Baum der Höhe n mit nur linken Kindern (ein Pfad). Sucht man nun nacheinander die Elemente $1, \ldots, n$ so ist, benutzt man Zig&Zig anstatt Zig-Zig, so verringert sich die Höhe immer nur um eins, und schlimmer, daß gesuchte Element ist immer ganz unten. Diese Folge von Suchen benötigt dann also $\Theta(n^2)$ Zeit.

Probiert man das gleiche Beispiel bei einem wirklichen Splaybaum, so halbiert sich die Höhe bereits in der ersten Suche.

Aufgabe T17

Beschreiben Sie einen Algorithmus, der mit Hilfe dynamischer Programmierung berechnet, wie viele verschiedene binäre Suchbäume, welche die Schlüssel $1, \ldots, n$ enthalten, existieren.

Lösungsvorschlag: Fixieren wir ein $n > 0$ und definieren wir t_n als die Anzahl der Suchbäume, welche n Schlüssel enthalten. Nehmen wir an, die wir kennen t_i für alle $i < n$ und wollen t_n bestimmen. Wir haben n Möglichkeiten, eine Wurzel zu wählen. Überlegen wir, was passiert wenn der k-te Schlüssel als Wurzel gewählt wurde: der linke Teilbaum kann jeder der t_{k-1} binären Suchbäume mit $k-1$ Schlüsseln sein, der reichte analog jeder der t_{n-k} binären Suchbäume mit $n-k$ Schlüsseln. Daraus ergbit sich die Rekursion

$$t_n = \begin{cases} \sum_{k=1}^{n} t_{k-1} \cdot t_{n-k} & n > 0 \\ 1 & n = 0 \end{cases}$$

Man beachte, daß der leere Baum durch t_0 gezählt wird.

Aufgabe T18

Verwenden Sie LuFGTI-Schnipsel, um zwei Zufallszahlen a, b mit $1 \leq a \leq 6$ und $0 \leq b \leq 6$ zu bestimmen. Verwenden Sie diese persönlichen Glückszahlen, um die untenstehenden Zahlen mit der Funktion $h(x) = (ax + b) \bmod 7$ zu hashen.

x	$a \cdot x$	$a \cdot x + b$	$(a \cdot x + b) \bmod 7$
2			
4			
5			
7			
10			
12			
100			
1000			

Tragen Sie die Zahlen anschließend in eine Hashtabelle der Größe $m = 7$ ein. Verwenden Sie dabei Listen, um Kollisionen zu behandeln. Ermitteln Sie, wie viele Listenvergleiche eine (erfolglose) Suche nach dem Element 3 benötigt. Was ist der Durchschnitt in der Übungsgruppe? Was ist der Erwartungswert?

Lösungsvorschlag: Hinterhältigerweise haben wir in diesem Beispiel Elemente gewählt, die nicht im zulässigen Intervall von $0...p-1$ liegen. Für dieses Intervall wäre die erwartete Anzahl von Listenvergleichen $O(\alpha) = O(n/m)$.

Hier aber sollte sich, falls sich niemand verrechnet hat, genau eine Kollision für jedes mögliche Paar a, b ergeben. Dazu überlegen wir uns, daß kein Element zwischen 0 und $p - 1 = 6$ mit dem Element 3 kollidieren kann (ausgenommmen die 3 selbst, natürlich), denn $\mathbb{Z}/7\mathbb{Z}$ ist ein Körper und die Funktionen $(ax + b) \bmod 7$ sind darauf bijektiv (siehe Vorlesungsfolien).

Welche Elemente ≥ 7 aber kollidieren mit der 3? Ene schnelle Rechnung ergibt

$$3a + b \equiv_7 ax + b \Leftrightarrow 3a \equiv_7 ax \Leftrightarrow 3 \equiv_7 x$$

Damit kollidieren alle Elemente der Restklasse 3 miteinander! In unserem Beispiel ist das nur das Element 10, daher rührt also die einzige Kollision.

Aufgabe H12 (10 Punkte)

Implementieren Sie einen Algorithmus, der für natürliche Zahlen n, h mit $h \leq n$ bestimmt, wie viele binäre Suchbäume der Höhe h mit den Schlüsseln $1, \ldots, n$ existieren. Bestimmen Sie für $0 \leq n \leq 20$ jeweils die *durchschnittliche* Höhe und plotten Sie das Ergebnis in einer geeigneten Form oder fertigen Sie eine Tabelle an. Stellen Sie eine Vermutung über den Verlauf der so dargestellten Funktion auf.

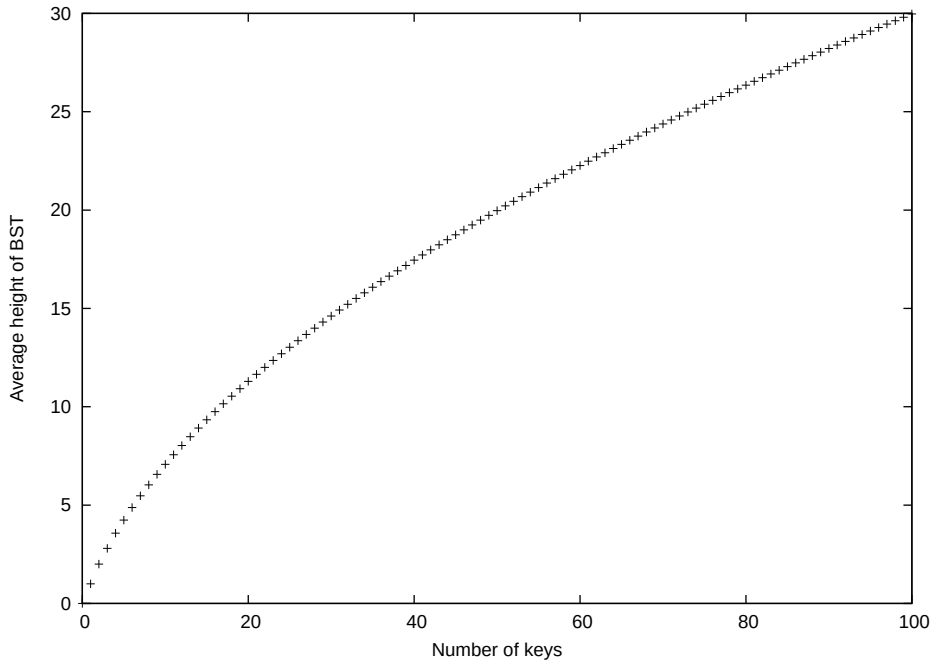
In der Vorlesung wurde bewiesen, daß eine zufällige Einfügereihenfolge von n Schlüsseln einen Suchbaum mit erwarteter Höhe von $O(\log n)$ ergibt. Wie verträgt sich dieser Sachverhalt mit Ihrer Beobachtung?

Lösungsvorschlag: Wir vereinfachen uns die Aufgabe, indem wir $t_{n,h}$ als die Anzahl der binären Suchbäume mit n Schlüsseln mit *höchstens* Höhe h definieren. Die Differenz $t_{n,h} - t_{n,h-1}$ ist dann gerade die gesuchte Anzahl von Bäumen.

Eine mögliche Rekursionsgleichung für diese Umformulierung lautet

$$t_{n,h} = \begin{cases} \sum_{k=1}^{n} t_{k-1,h-1} \cdot t_{n-k,h-1} & n > 0 \\ 1 & n = 0 \end{cases}$$

Plottet man die durchschnittliche Höhe der Bäume gegen die Anzahl der Schlüssel, dann erhält man eine Funktion, die der Wurzelfunktion ähnelt (tatsächlich liegt die Funktion in $\Theta(\sqrt{n})$, der Beweis ist aber kompliziert).



Ein zufällig gewählter Binärbaum hat also im Erwartungswert eine Höhe von $\Theta(\sqrt{n})$—wählen wir hingegen zufällig eine Einfügereihenfolge (eine Permutation) der Schlüssel, so ergibt sich ein Binärbaum mit erwarteter Höhe von $O(\log n)$. Daraus schließen wir, daß viele Einfügereihenfolgen den gleichen Binärbaum ergebeben und daß Bäume mit kleiner Höhe durch mehr solche Reihenfolgen erzeugt werden als Bäume mit großer Höhe.

Eine Implementation mit dynamischer Programmierung befindet sich auf der Vorlesungsseite (es benötigt die GNU MP library).

Aufgabe H13 (10 Punkte)

Die Größe einer Hashtabelle wird gemäß folgender Strategie angepaßt:

- Falls sie mehr als 10 Elemente enthält und der Lastfaktor mindestens drei ist, wird die Hashtabelle durch eine Tabelle dreifacher Größe ersetzt.
- Falls sie mehr als 10 Elemente enthält und der Lastfaktor höchstens 1/3 ist, wird die Hashtabelle durch eine Tabelle ersetzt, deren Größe nur 1/3 der aktuellen Tabellengröße entspricht.
- Bei höchstens 10 Elementen wird eine Tabelle der Größe 30 verwendet.

Finden Sie eine geeignete Potentialfunktion, bezüglich welcher die amortisierten Kosten des Einfügens und des Löschens konstant sind. Beweisen Sie dies.

Lösungsvorschlag:

Wir verwenden als Potentialfunktion die Anzahl der Operationen seit dem letzten Rehash mal einer Konstante c. Sei α der aktuelle Lastfaktor.

Falls kein Rehash stattfindet, dann sind die tatsächlichen Kosten im Erwartungswert durch $O(1+\alpha)$ beschränkt, da die Anzahl der Vergleiche im Durchschnitt höchstens $1+\alpha$ ist. Außerdem steigt das Potential um c, so daß die amortisierten Kosten höchstens $O(1+\alpha+c)$ betragen. Da α niemals größer als drei ist, sind die amortisierten Kosten also höchstens $O(1)$.

Im Falle eines Rehash müssen alle Elemente der Hashtabelle durchgegangen werden, was $O(n+m)$ Zeit benötigt, falls n die Anzahl der gespeicherten Elemente und m die augenblickliche Größe der Hashtabelle ist. Für das Speichern in die neue Hashtabelle wird wiederum $O(n)$ Zeit benötigt, insgesamt also $O(n)$, da $m = O(n)$.

Seit dem letzten Rehash wurden aber mindestens $2n$ Elemente gelöscht oder $2n/3$ Elemente eingefügt. Die Potentialfunktion fällt auf 0, nimmt also um mindestens $c2n/3$ ab.

Die amortisierten Kosten sind daher durch $O(n) - c2n/3$ beschränkt, wobei die Konstante im $O(n)$ nicht von c abhängt. Wenn wir c groß genug wählen, sinken die amortisierten Kosten sogar unter null.

Aufgabe H14 (10 Punkte)

Aus seiner Abneigung gegenüber Listen mit doppelten Einträgen heraus hat Felix beschlossen, einen Webservice anzubieten, um eben solche zu erkennen. Der Kernalgorithmus ist (verkürzt) auf der rechten Seite abgebildet: mit Hilfe einer Hashtabelle wird nach doppelten Einträgen gesucht.

Leider hat er sich keine Gedanken über die Sicherheit des LuFGTI-Webservers gemacht. Beweisen Sie ihm, daß sein Algorithmus bereits bei einer Liste mit höchstens 1000 Elementen mehr als 400000 Vergleiche benötigen kann und damit verwundbar für eine Denial-of-Service-Attacke ist!

Beschreiben Sie ihr Vorgehen und geben Sie eventuell benutzte Programme an.

```
public static boolean hasDoublet(int n, int[] list) {
    int m = 2 * n;
    int c = (int) (Math.random() * 1000);
    ArrayList<ArrayList<Integer>> buckets =
            new ArrayList<ArrayList<Integer>>();
    for (int i = 0; i < m; ++i)
        buckets.add(new ArrayList<Integer>());
    for (int i = 0; i < n; ++i) {
        int key = hash(list[i], m, c);
        for (Integer element : buckets.get(key)) {
            if (element == list[i])
                return true;
        }
        buckets.get(key).add(list[i]);
    }
    return false;
}

public static int hash(int x, int m, int c) {
    int r = x % 23;
    r = r * r * r * r;
    return ((x % m) * (x % m) + 3 * x + r + c) % m;
}
```

Auf `http://tcs.rwth-aachen.de/lehre/DA/SS2011/` finden Sie das Programm sowohl in Java als auch in C implementiert. Es gibt praktischerweise die Anzahl der benötigten Vergleiche direkt aus.

Lösungsvorschlag:
Das folgende Programm benutzt eine schlichte Brute-Force-Suche, um 1000 Zahlen mit dem gleichen Hashwert zu finden. Dazu sind zwei Beobachtungen essentiell: zum einen ist die Tabellengröße m von n abhängig und damit bekannt. Zum anderen bewirkt die Zufallszahl c nur eine Verschiebung der Hashwerte, es gilt

$$\text{hash}(x, m, c_1) = \text{hash}(y, m, c_1) \Leftrightarrow \text{hash}(x, m, c_2) = \text{hash}(y, m, c_2)$$

und damit bleiben Kollision trotz verschiedener Werte für c erhalten!

```
import java.io.*;
import java.util.ArrayList;

public class HashDoSAttack {

    public static void main(String[ ] args) throws IOException {
        ArrayList<Integer> res = new ArrayList<Integer>();
        int c = 0;
        while(res.size() < 1000) {
            while(hash(c, 2 * 1000) ≠ 0) c++;
            res.add(c);
            c++;
        }

        StringBuilder s = new StringBuilder("1000 ");
        for(Integer i : res) {
            s.append(i).append(" ");
        }

        System.out.println(s);
    }

    public static int hash(int x, int m) {
        int r = x % 23;
        return ((x % m) * (x % m) + 3 * x + r * r * r * r) % m;
    }
}
```

Die mit diesem Programm generierte Liste benötigt 500500 Operationen.

RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Übungsblatt 6
Emmes—Langer—Reidl—Rossmanith 16.05.2011

Übung zur Vorlesung Datenstrukturen und Algorithmen

Aufgabe T19

Überlegen Sie, wie sich mit Hilfe eines einzigen 32-Bit Integers Teilmengen von $\{0, \ldots, 30\}$ darstellen lassen und geben Sie an, wie die folgenden Operationen implementiert werden können: (1) Einfügen, Löschen und Enthaltensein eines Elements, und (2) Vereinigung und Schnitt zweier Mengen.

Lösungsvorschlag:

```
class MyBitSet {
    int content = 0;
    void add (int i) {content |= 1 << i;}
    void remove (int i) {content &= ~(1 << i);}
    boolean contains (int i) {return (content & (1 << i)) ≠ 0;}
    void union (MyBitSet b) {content |= b.content;}
    void intersection (MyBitSet b) {content &= b.content;}
    void difference (MyBitSet b) {content &= ~b.content;}
    boolean isSubsetOf (MyBitSet b) {return (content & ~b.content) ≡ 0;}
}
```

Aufgabe T20

Eine Skip-List hat einen Head-Knoten, der eine bestimmte Höhe besitzt. Soll ein Element mit einer größeren Höhe in die Liste eingefügt werden, so muss der Head-Knoten vergrößert werden—der in der Vorlesung präsentierte Algorithmus zum Löschen von Elementen verändert die Höhe des Head-Knoten allerdings nie. Kann dies zu einem schlechteren Laufzeitverhalten führen, als eine Implementierung die beim Löschen den Head-Knoten verkleinert?

Lösungsvorschlag:

Ja. Wenn man n mal hintereinander ein Element in eine leere Liste einfügt und danach wieder löscht, liegt die erwartete Höhe des Head-Knoten bei $\log n$. Wird nach jedem Einfügen ein Element in der Liste gesucht, welches größer als das Vorhandene ist, benötigt man hierfür asymptotisch $\sum_{i=1}^{n} \log i$ viele Vergleiche, was wir wegen

$$\lim_{n\to\infty} \frac{\sum_{i=1}^{n} \log i}{n \log n} \underbrace{=}_{\text{L'Hôpital}} \lim_{n\to\infty} \frac{\sum_{i=1}^{n} \frac{1}{i}}{\log n + 1} = \lim_{n\to\infty} \frac{H_n}{\log n + 1} \underbrace{=}_{\text{Siehe H2}} \Theta(1)$$

mit $\Theta(n \log n)$ abschätzen können.

Verwendet man hingegen einen Algorithmus, der nach jedem Löschen die Höhe des Head-Knoten minimiert, liegt die erwartete Höhe beim folgenden Suchen in der einelementigen Liste bei 2. Somit benötigt man insgesamt nur linear viele Vergleiche.

Aufgabe T21

In der Vorlesung wurde Quicksort auch iterativ mit Hilfe eines expliziten Stacks implementiert. Da Speicherverbrauch immer ein wichtiger Faktor ist, sind wir an der maximalen Höhe dieses Stacks interessiert: Finden Sie ein Beispiel, in welchem die gegebene Quicksort-Implementation $\Omega(n)$ Paare gleichzeitig im Stack speichert. Überlegen Sie dann, wie der Algorithmus abgeändert werden kann, um diesen schmerzhaften Speicherverbrauch deutlich zu senken.

Aus Effizienzgründen bestimmt die vorliegende Implementation zunächst das minimale Element des Eingabearrays und vertauscht es mit dem ersten Element desselben, die verbleibenden Elemente werden dann wie gehabt sortiert.

```
public void quicksort() {
  Stack⟨Pair⟩ stack = new Stack⟨Pair⟩();
  stack.push(new Pair(1, size − 1));
  int min = 0;
  for(int i = 1; i < size; i++)
    if(less(i, min)) min = i;
  D t = get(0); set(0, min); set(min, t);
  while(!stack.isempty()) {
    Pair p = stack.pop();
    int l = p.first(), r = p.second();
    int i = l − 1, j = r, pivot = j; D t;
    do {
      do {i++;} while(less(i, pivot));
      do {j−−;} while(less(pivot, j));
      t = get(i); set(i, get(j)); set(j, t);
    } while(i < j);
    set(j, get(i));
    if(r − i > 1) stack.push(new Pair(i + 1, r));
    if(i − l > 1) stack.push(new Pair(l, i − 1));
  }
}
```

Die Analyse von Quicksort setzt jedoch voraus, daß jede mögliche Permutation der zu sortierenden Schlüssel gleich wahrscheinlich ist. Sind nach der obigen Veränderung der Eingabe alle Permutationen der verbleibenden Elemente immer noch gleich wahrscheinlich?

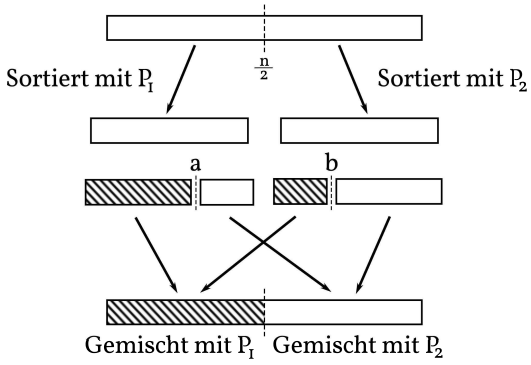
Lösungsvorschlag:

Das oberste Element des Stacks wird sofort am Anfang der Schleife entfernt, interessant ist also, das erste der beiden eingefügten Intervalle. Bringen wir den Algorithmus dazu, als erstes Element ein sehr kleines Intervall (etwa der Größe eins) auf den Stack zu legen und dann gezwungenermaßen ein sehr großes Intervall als zweites, so benötigt Quicksort $\Omega(n)$ Iterationen der äußeren Schleife—eine Beispielinstanz dafür wäre schlicht ein bereits aufsteigend sortiertes Array. Da nun in jeder Iteration ein Element auf dem Stack verbleibt, erreicht der Stack eine Größe von $\Omega(n)$.

Dieses Verhalten kann verhindert werden, indem man immer das kürzere Intervall oben auf den Stack legt: so kann der Stack maximal eine Größe von $O(\log n)$ erreichen.

Nehmen wir o.b.d.A. an, die Eingabe bestehe aus einer beliebigen Permutation der Elemente $1 \dots n$ und alle diese Permutationen haben die gleiche Wahrscheinlichkeit. Ein einfaches Zählargument zeigt, daß die anfängliche Vertauschung Permutationen auf den Elementen $2 \dots n$ erzeugt, die wieder alle gleich wahrscheinlich sind: für jede der $(n-1)!$ vielen Permutationen können n verschiedene Permutation der Elemente $1 \dots n$ erzeugt werden, indem ein Element durch die 1 ersetzt und anschließend den anderen vorangestellt wird. Da dies niemals zwei gleiche Permutationen erzeugt, erhalten wir $n(n-1)! = n!$ viele Permutationen auf n Elementen—umgekehrt ist damit die Wahrscheinlichkeit, eine konkrete Permutation der Elemente $2 \dots n$ zu erhalten $n\frac{1}{n!} = \frac{1}{(n-1)!}$

Aufgabe H15 (15 Punkte)



Betrachten wir die unten beschriebene, parallelisierte Variante von Mergesort. Wir setzen zwei Prozessoren P_1, P_2 ein, um jeweils eine Hälfte der Eingabe (seriell) zu sortieren. Sei zu diesem Zweck n die Länge des Eingabearrays; P_1 sortiere dann den Teil $0 \dots m-1$ und P_2 den Teil $m \dots n-1$ mit $m = \lceil n/2 \rceil$.
Jetzt soll auch das anschließende Mischen dieser nun geordneten Teilarrays parallel stattfinden: dazu bestimme man zwei Zahlen a, b mit $0 \leq a \leq m-1$ und $m \leq b \leq n-1$ so, daß der angegebenen Algorithmus korrekt sortiert und $a+b = n \pm 1$ gilt. Die von Ihnen vorgeschlagene Lösung sollte höchstens $O\big((\log n)^2\big)$ Schritte benötigen.

1: **sort**$(A[0 \dots m-1])$ *on* P_1 || **sort**$(A[0 \dots m-1])$ *on* P_2
2: *Bestimme a und b*
3: $c = a + b - m + 1$
4: $T[0 \dots n-1] := A[0 \dots n-1]$
5: **merge**$(T[0 \dots a], T[m \dots b-1])$ *into* $A[0 \dots c-1]$ *on* P_1 ||
merge$(T[a+1 \dots m-1], T[b, n-1])$ *into* $A[c \dots m-1]$ *on* P_2

Ergeizige können versuchen, es in nur $O(\log n)$ Schritten zu schaffen.

Lösungsvorschlag:

```
int lmid, rmid;
int lleft = 0;
int lright = half − 1;
int rleft = half;
int rright = n − 1;
int side = 1;
while(lleft + 1 < lright && rleft + 1 < rright) {
    if(side > 0) {
        lmid = lleft + ((lright − lleft) >> 1);
        int index = find(array[lmid], array, rleft, rright);
        if (lmid + index ≥ n)
            lright = lmid;
        else
            lleft = lmid;
    } else {
        rmid = rleft + ((rright − rleft) >> 1);
        int index = find(array[rmid], array, lleft, lright);
        if (rmid + index ≥ n)
            rright = rmid;
        else
            rleft = rmid;
    }
    side = −side;
}
lmid = lleft + ((lright − lleft) >> 1);
rmid = find(array[lmid], array, rleft, rright);
```

Aufgabe H16 (5 Punkte)
Ändern Sie den Algorithmus der Vorlesung zum Löschen von Elementen aus einer Skip-Liste so ab, daß er den Head-Knoten, wenn möglich, verkleinert.

Lösungsvorschlag:

```
public void reduceHead() {
    Array⟨Node⟩ old = head.succ;
    int i = head.succ.size();
    while(i-- ≥ 0) {
        if(old.get(i) ≠ tail)
            break;
    }

    head.succ = new Array⟨Node⟩();
    for(int j = 0; j ≤ i; j++){
        head.succ.set(j, old.get(j));
    }
}
```

Aufgabe H17 (10 Punkte)
Die Funktionen *puzzle* und *riddle* bekommen einen Integer-Wert als Argument, der wie in Aufgabe T19 eine Teilmenge von $\{0, \ldots, 30\}$ beschreibt. Suchen Sie sich eine der beiden Funktionen aus! Welche Operation auf Bitsets implementiert diese Funktion? Beschreiben Sie, wie sie funktioniert.

Bedenken Sie bei der Funktion *puzzle*, daß $(5)_{16} = (0101)_2$, $(A)_{16} = (1010)_2$, $(3)_{16} = (0011)_2$, $(C)_{16} = (1100)_2$, $(F)_{16} = (1111)_2$ und $(0)_{16} = (0000)_2$ gilt.

```
int puzzle(int m) {
    m = (m & 0x55555555)
     +((m & 0xAAAAAAAA) >> 1);
    m = (m & 0x33333333)
     +((m & 0xCCCCCCCC) >> 2);
    m = (m & 0x0F0F0F0F)
     +((m & 0xF0F0F0F0) >> 4);
    m = m + (m >> 8);
    m = (m + (m >> 16)) & 31;
    return m;
}
```

```
void riddle(int m) {
    int z = 0;
    int cp = ~m + 1;
    do {
        print(z);
        z = (z + cp) & m;
    } while (z ≠ 0);
}

void print(int c) {
    String s = "";
    for (int i = 0; i ≤ 31; ++i) {
        s = (c & 1) + s;
        c = c >> 1;
    }
    System.out.println(s);
}
```

Lösungsvorschlag:
Die Methode *puzzle* berechnet die Kardinalität der durch m beschriebenen Menge, welche im folgenden mit M bezeichnet sei. Dazu überlege man sich, daß nach der ersten Addition m die Kardinalität der Mengen

$$M \cap \{0,1\}, M \cap \{2,3\}, \ldots M \cap \{28,29\}, M \cap \{30\}$$

enthält, und zwar jeweils in Bitgruppen der Größe zwei (so ist die Kardinalität z.B. der Menge $M \cap \{2,3\}$ an den Bits 2 und 3 abzulesen).
Die darauffolgende Addition ergibt dann die Kardinalitäten der Mengen

$$M \cap \{0,1,2,3\}, \ldots, M \cap \{24,25,26,27\}, M \cap \{28,29,30\}$$

jeweils in Bitgruppen der Größe vier.
Ab der vierten Addition ist ein Überlauf nicht mehr Möglich (die Kardinalität der bis dato gezählten Teilmengen ist höchstens 8 und benötigt damit höchstens vier Bit), weswegen auf die Maskierung verzichtet werden kann. Die letzte Maskierung mit 31 löscht einen eventuellen Überlauf auf das 32. Bit.

Die Methode *riddle* enumeriert alle Teilmengen der Menge M. Grundlegende ist dabei die Operation

$$\begin{aligned} z &\leftarrow (z + cp) \;\&\; m \Leftrightarrow \\ z &\leftarrow (z + 1 + \sim m) \;\&\; m \end{aligned}$$

Zunächst überlege man sich, wie z sich verhält wenn $\sim m = 0$ ist: dann wird in jedem Durchlauf der Schleife z um eins erhöht und enumeriert damit alle Teilmengen von $\{0,\ldots,30\}$. Nun sei $\sim m \neq 0$, es gibt also Elemente aus $\{0,\ldots,30\}$, die *nicht* in M enthalten sind.

Der Einfachheit halber werden wir das i-te Bit der Zahl z mit $z[i]$ bezeichnen (für $1 \leq i \leq 30$). Nehmen wir an, z repräsentiere eine Teilmenge von M dies ist zu Beginn mit $z = 0$ gegeben). Repräsentiert $z + 1$ ebenfalls eine Teilmenge von M, so ist

$$z \leftarrow (z + 1 + \sim m) \ \& \ m = (z + 1) \ \& \ m = z + 1$$

denn es gilt $(z+1)[i] = (\sim m)[i], 0 \leq i \leq 30$ per Annahme—die anschließende Maskierung mit m ändert dann natürlich auch nichts. Nehmen wir also im folgenden an, daß $z + 1$ keine Teilmenge von M repräsentiert. Da dies noch für z galt, muß das niederwertigste Bit von $z + 1$ einen Index $i \notin M$ haben. Durch die Addition von $z + 1$ mit $\sim m$ wird diese Bit auf jeden Fall gelöscht, denn $(\sim m)[i] = 1$, ebenso wie alle darauffolgenden Indizes j für die $(\sim m)[j] = 1$ gilt—der Überlauf 'stoppt' bei dem ersten Index $j > i$ mit $(\sim m)[j] = 0$, was gleichbedeutend mit $m[j] = 1$ ist. Damit werden alle Bits, die nicht in m gesetzt sind und somit in keiner Teilmengenräpresentation auftauchen können, bei dieser Addition übersprungen. Die Anschließende Maskierung mit m entfernt wiederum alle so nicht gelöschten Bits.

Übung zur Vorlesung Datenstrukturen und Algorithmen

Aufgabe T22

Benutzen Sie Quickselect, um in dem folgenden Array das fünftgrößte Element zu bestimmen.

$$7, 3, 8, 5, 2, 10, 15, 3, 3, 4, 8, 12, 1, 6$$

Lösungsvorschlag:

$$\underset{\substack{L\\\uparrow}}{7}, 3, 8, 5, 2, 10, 15, 3, 3, 4, 8, 12, 1, \underset{R}{6}, \ ^{\downarrow}$$

$$\underset{L}{7}, 3, \underset{\uparrow}{6}, 5, 2, 10, 15, 3, 3, 4, 8, 12, 1, \overset{\downarrow}{\underset{R}{8}}$$

$$\underset{L}{7}, 3, 6, 5, 2, \underset{\uparrow}{1}, 15, 3, 3, 4, 8, 12, \overset{\downarrow}{10}, \underset{R}{8}$$

$$\underset{L}{7}, 3, 6, 5, 2, 1, \underset{\uparrow}{4}, 3, 3, \overset{\downarrow}{15}, 8, 12, 10, \underset{R}{8}$$

$$\underset{L}{7}, 3, 6, 5, 2, 1, 4, 3, \overset{\downarrow}{15}, \underset{\uparrow}{3}, 8, 12, 10, \underset{R}{8}$$

$$\underset{L}{3}, 3, 6, 5, 2, 1, 4, 3, \overset{\downarrow}{7}, \underset{\uparrow}{15}, 8, 12, 10, \underset{R}{8}$$

$$\underset{\substack{L\\\uparrow}}{3}, 3, 6, 5, 2, 1, 4, 3, \underset{R}{7}, \overset{\downarrow}{15}, 8, 12, 10, 8$$

$$\underset{L}{3}, \underset{\uparrow}{3}, 6, 5, 2, 1, 4, \overset{\downarrow}{3}, \underset{R}{7}, 15, 8, 12, 10, 8$$

$$\underset{L}{3}, \underset{\uparrow}{3}, 6, 5, 2, 1, 4, \overset{\downarrow}{3}, \underset{R}{7}, 15, 8, 12, 10, 8$$

$$\underset{L}{3}, 3, \underset{\uparrow}{1}, 5, 2, \overset{\downarrow}{6}, 4, 3, \underset{R}{7}, 15, 8, 12, 10, 8$$

$$\underset{L}{3}, 3, 1, \overset{\downarrow}{5}, \underset{\uparrow}{2}, 6, 4, 3, \underset{R}{7}, 15, 8, 12, 10, 8$$

$$\underset{L}{2}, 3, 1, \overset{\downarrow}{3}, \underset{\uparrow}{5}, 6, 4, 3, \underset{R}{7}, 15, 8, 12, 10, 8$$

$$2, 3, 1, \underset{\substack{L\\\uparrow}}{3}, 5, 6, 4, 3, \underset{R}{7}, \overset{\downarrow}{15}, 8, 12, 10, 8$$

$$2, 3, 1, \underset{L}{3}, \underset{\uparrow}{3}, 6, 4, \overset{\downarrow}{5}, \underset{R}{7}, 15, 8, 12, 10, 8$$

$$2, 3, 1, \underset{L}{3}, \overset{\downarrow}{6}, \underset{\uparrow}{3}, 4, 5, \underset{R}{7}, 15, 8, 12, 10, 8$$

$$2, 3, 1, \underset{L}{3}, \overset{\downarrow}{3}, \underset{\uparrow}{6}, 4, 5, \underset{R}{7}, 15, 8, 12, 10, 8$$

Im letzten Schritt gilt $k = r$ und damit ist 3 das fünftgrößte Element.

Aufgabe T23

Benutzen Sie Heapsort, um das folgende Array zu sortieren:

$$143, 6, 8, 32, 76, 90, 1, 55$$

Lösungsvorschlag:

Der Aufgebaute Heap hat die folgende Struktur:

$$61, 32, 6, 55, 76, 90, 8, 143$$

Nach dem jeweiligen Entfernen der Wurzel und der Wiederherstellung der Heapeigenschaft ergeben sich folgende Heaps:

$$\begin{array}{rl}
(1) & 6, 32, 8, 55, 76, 90, 143 \\
(6) & 8, 32, 90, 55, 76, 143 \\
(8) & 32, 55, 90, 143, 70 \\
(32) & 55, 70, 90, 143 \\
(55) & 70, 143, 90 \\
(70) & 90, 143 \\
(90) & 143 \\
(143) &
\end{array}$$

Aufgabe T24

Für die Vorlesung war noch zu zeigen, daß aus der Rekursionsgleichung

$$C_n = n + 1 + \frac{2}{n} \sum_{i=\lceil n/2 \rceil}^{n-1} C_i$$

mit $C_0 = C_1 = 0$ folgt, daß $C_n \leq 4n$. Beweisen Sie dies mittels Induktion!

Lösungsvorschlag:

Der Induktionsanfang ist durch C_0, C_1 gegeben. Nehmen wir an, $C_{n-1} \leq 4(n-1)$ sei bereits bewiesen. Dann ist

$$\begin{aligned}
C_n &\leq n + 1 + \frac{2}{n} \sum_{i=\lceil n/2 \rceil}^{n-1} C_i \underset{IV}{\leq} n + 1 + \frac{2}{n} \sum_{i=\lceil n/2 \rceil}^{n-1} 4i \\
&\leq n + 1 + \frac{2}{n} \sum_{i=1}^{n-1} 4i - \frac{2}{n} \sum_{i=1}^{n/2-1} 4i \leq n + 1 + \frac{8}{n} \frac{(n-1)n}{2} - \frac{8}{n} \frac{(n/2)(n/2-1)}{2} \\
&= n + 1 + 4n - 4 - 2(n/2 - 1) = 5n - 3 - n - 1 \\
&\leq 4n
\end{aligned}$$

Aufgabe H18 (5 Punkte)

Für die Vorlesung war noch zu zeigen, daß aus der Rekursionsgleichung

$$\begin{aligned}
&C_n \leq n + 1 + C_{\lfloor n/5 \rfloor} + C_{\lfloor 3n/4 \rfloor} \text{ für } n > 30 \\
&C_n = O(1) \text{ für } n \leq 30
\end{aligned}$$

folgt, daß $C_n = O(n)$. Beweisen Sie dies per Induktion!

Lösungsvorschlag:
Wir zeigen $C_n \leq cn$ für eine geeignete Konstante $c > 0$. Der Induktionsanfang ist wegen $C_n = O(1)$ für $n \leq 30$ offenbar gegeben. Sei daher $n > 30$ und $C_{n'} \leq cn'$ für alle $n' < n$. Dann ist

$$\begin{aligned} C_n &\leq n + 1 + C_{\lfloor n/5 \rfloor} + C_{\lfloor 3n/4 \rfloor} \\ &\leq n + 1 + c\lfloor n/5 \rfloor + c\lfloor 3n/4 \rfloor \\ &= n + 1 + c(\lfloor n/5 \rfloor + \lfloor 3n/4 \rfloor) \\ &\leq n + 1 + c(n/5 + 3n/4) \\ &= n + 1 + c(4n + 15n)/20 \\ &= n + 1 + cn19/20 \leq cn, \end{aligned}$$

falls $c > 21$.

Aufgabe H19 (15 Punkte)
Die Klausur zur Veranstaltung "Dadaismus für Informatiker" (DafI) soll ausgewertet werden, denn die Teilnehmer sollen ihrer Note nach belohnt werden (die besten Studenten erwartet ein orangefarbener Kaktus). Die Notenskala für diese Klausur lautet, von der besten zur schlechtesten Note:

Spargel, Hüfte, Gabel, Baum, Kaminrot, Schnupfen, Heimweh

Nun müssen die Teilnehmerdaten nach Noten sortiert werden; aus Kostengründen und akutem Speichermangel in der Fakultät für Moderne Kunst muß dieses Sortieren (1) in Linearzeit geschehen und (2) In-Place funktionieren.
Entwerfen Sie einen Sortieralgorithmus, der den oben genannten Anforderungen entspricht. Nehmen Sie an, daß die Daten in einem Array vorliegen und daß die Relationen $<, >$ und $=$ für die Noten implementiert sind.

Lösungsvorschlag:
Der folgende Algorithmus verwendet für die Noten natürliche Zahlen $0, \ldots, 6$, indem er vorher jeder der obigen Noten eine Zahl in aufsteigender Reihenfolge, die der Notenfolge entspricht, zugeordnet hat.

Die grundlegende Idee ist es, den Vertauschungsalgorithmus von Quicksort zu benutzen um zunächst alle Datensätze mit der besten Note (Spargel, hier durch eine 0 repräsentiert) an den Anfang des Arrays zu setzten. Danach wird im verbleibenden Array—angefangen an der Stelle $left - 1$, an welcher nun der letzte Datensatz mit der Note Spargel steht— Verfahren für die Nächsthöhere Note (Hüfte) angewandt usw.

Analog zu Quicksort benötigen wir ein Guard-Element am Ende des Arrays, um den linken Zeiger davor zu bewahren, aus dem Array zu laufen.

Jeder Durchlauf der inneren Schleife benötigt lineare Zeit, die äußere Schleife wird siebenmal durchlaufen—insgesamt also $O(n)$ Schritte. Offensichtlich arbeitet der Algorithmus In-Place.

```
int left = −1;
int right = n + 1;
int t;
for(int c = 0; c ≤ 6; c++) {
    right = n;
    do {
        do{left++; } while(array[left] ≡ c);
        do{right−−; } while(array[right] ≠ c);
        t = array[left];
        array[left] = array[right];
        array[right] = t;
    } while(left < right);
    t = array[left];
    array[left] = array[right];
    array[right] = t;
    left−−;
}
```

RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Übungsblatt 8
Emmes—Langer—Reidl—Rossmanith 30.05.2011

Übung zur Vorlesung Datenstrukturen und Algorithmen

Aufgabe T25

	Quicksort	Heapsort	Mergesort	Insertion-Sort	Straight-Radix	Radix-Exchange
in-place?						
stabil?						
Laufzeit (worst-case)						
Laufzeit (Durchschnitt)						
vergleichsbasiert?						

Beantworten Sie die Fragen für alle Sortierverfahren. Gehen Sie davon aus, daß ein Vergleich in konstanter Zeit durchgeführt wird und die Anzahl der zu sortierenden Elemente n beträgt. Für Laufzeiten tragen Sie eine Funktion $f(n)$ in die Tabelle ein, um eine Laufzeit von $O(f(n))$ auszudrücken.

Lösungsvorschlag:

	Quicksort	Heapsort	Mergesort	Insertion-Sort	Straight-Radix	Radix-Exchange
in-place?	??	J	N	J	N	??
stabil?	N	N	J	J	J	N
Laufzeit (worst-case)	n^2	$n \log n$	$n \log n$	n^2	nw	nw
Laufzeit (Durchschnitt)	$n \log n$	$n \log n$	$n \log n$	n^2	nw	nw
vergleichsbasiert?	J	J	J	J	N	N

Bei den Radix-Sortierverfahren bezeichne w die Wortlänge. Quicksort und Radix-Exchange-Sort sind wegen des benötigten Stacks nicht in-place. Beide lassen sich jedoch so implementieren, daß der Stack in rekursiven Funktionsaufrufen „versteckt“ wird, während jeder einzelne Aufruf nur konstant viel Platz benötigt.

Aufgabe T26

	Liste	Sortierte Liste / Array	Sortiertes Array / Trie	Binärer Suchbaum / AVL-Baum / Splay-Tree	Treap	Skip-List	Hashtabelle	Min-Heap
Randomisiert?								
Einfügen								
Suchen								
Löschen								
Minimum								
Maximum								
Sort. Ausgeben								

Beantworten Sie die Fragen für alle Datenstrukturen bzw. geben Sie eine obere Schranke für den Aufwand an. Gehen Sie davon aus, daß die Anzahl der enthaltenden Elemente n beträgt. Für Laufzeiten tragen Sie eine Funktion $f(n)$ in die Tabelle ein, um eine Laufzeit von $O(f(n))$ auszudrücken. Bei randomisierten Datenstrukturen ist die erwartete obere Schranke gemeint, bei amortisierten Analysen die amortisierte Laufzeit.

Lösungsvorschlag:

	Liste	Sortierte Liste / Array	Sortiertes Array / Trie	Binärer Suchbaum / AVL-Baum	Splay-Tree	Treap	Skip-List	Hashtabelle	Min-Heap
Rand?	N	N	N	N	N	N	J	J	J	N
Einfügen	n	n	1	n	$\|x\|$	n	$\log n$	$\log n$	$\log n$	$\log n$	1	n
Suchen	n	n	n	$\log n$	$\|x\|$	n	$\log n$	$\log n$	$\log n$	$\log n$	1	n
Löschen	n	n	n	n	$\|x\|$	n	$\log n$	$\log n$	$\log n$	$\log n$	1	n
Min	n	1	n	1	$\|x\|$	n	$\log n$	$\log n$	$\log n$	1	n	1
Max	n	1	n	1	$\|x\|$	n	$\log n$	$\log n$	$\log n$	$\log n$	n	n
So. Aus	$n \log n$	n	$n \log n$	n	$\sum \|x\|$	n	n	n	$n \log n$	n	$n \log n$	$n \log n$

$|x|$ ist die Länge des aktuellen Strings.

Aufgabe T27

Entwickeln Sie eine Möglichkeit, die Adjazenzmatrix eines ungerichteten Graphens mit n Knoten in höchstens $n(n-1)/2$ Einträgen zu speichern, so daß es trotzdem in $O(1)$ Schritten möglich ist zu entscheiden, ob zwei Knoten i, j adjazent sind.

Lösungsvorschlag:

Da der Graph ungerichtet ist, können Anfragen zur Adjazenz zweier Knoten i, j immer auf den Fall $i < j$ zurückgeführt werden. Den Fall $i = j$ beantworten wir immer mit "nein". Dieses entspricht genau der Darstellung in einer Adjazenzmatrix, in der nur die Dreiecksmatrix unterhalb bzw. oberhalb der Diagonalen verwendet werden. Man beachte,

daß diese Dreiecksmatrix genau $n(n-1)/2$ Einträge hat, die wir nun geeignet speichern möchten.

Wir verwenden dazu ein Array, in der für die Knoten $1 \leq j \leq n$ in aufsteigender Reihenfolge die entsprechenden Zeilen der unteren Dreiecksmatrix einfach nacheinander gespeichert werden. Wir müssen nur noch effizient herausfinden können, an welcher Stelle im Array die Zeile für den Knoten j zu finden ist. Dazu beobachten wir, daß für den ersten Knoten keine Einträge, für den zweiten Knoten einen Eintrag (Adjazenz zum ersten Knoten), für den dritten Knoten zwei Einträge usw., und für den $j$ten Knoten $j-1$ Einträge gespeichert werden. Für n Knoten ergeben sich also insgesamt $\sum_{j=1}^{n}(j-1) = n(n-1)/2$ nötige Einträge. Außerdem erkennen wir direkt, daß der Beginn der Zeile für den $j$ten Knoten nun an der Position $p(j) := (j-1)(j-2)/2$ im Array zu finden ist. Der der Eintrag zur Adjazenz der Knoten i und j für $i < j$ befindet sich daher an Position $p(j) + i - 1 = (j-1)(j-2)/2 + i - 1$ im Array. Diese Position ist durch zwei Multiplikationen und eine Addition also in konstanter Zeit zu bestimmen.

Aufgabe H20 (10 Punkte)

Betrachten Sie wieder die Darstellung der Adjazenzmatrix eines ungerichteten Graphens mit n Knoten in einem Array aus Tutoraufgabe T26. Wir möchten basierend auf dieser Darstellung nun möglichst effizient die Nachbarschaft eines Knotens bestimmen, d.h. die Menge $N(j)$ der Knoten, zu denen der Knoten j adjazent ist. Beispielsweise möchte man oft wissen, wie viele Nachbarn der Knoten j hat (dieses nennt man den Grad des Knotens j).

Entwickeln Sie ein Verfahren, welches anhand des o.g. Arrays die Nachbarschaft eines Knotens in $\Theta(n)$ Schritten berechnet, und dabei nur eine *konstante* Anzahl an Multiplikationen verwendet.

Lösungsvorschlag:

Sei wie oben $p\colon j \mapsto (j-1)(j-2)/2$ und sei j der Knoten, dessen Nachbarschaft wir ermitteln möchten. Zunächst ist festzustellen, daß wir für jedes $1 \leq i \leq n$ mit $i \neq j$ überprüfen müssen, ob i und j adjazent sind. Insbesondere müssen wir für $i < j$ die Positionen $p(j) + i - 1$ des Arrays besuchen und für $i > j$ die Positionen $p(i) + j - 1$.

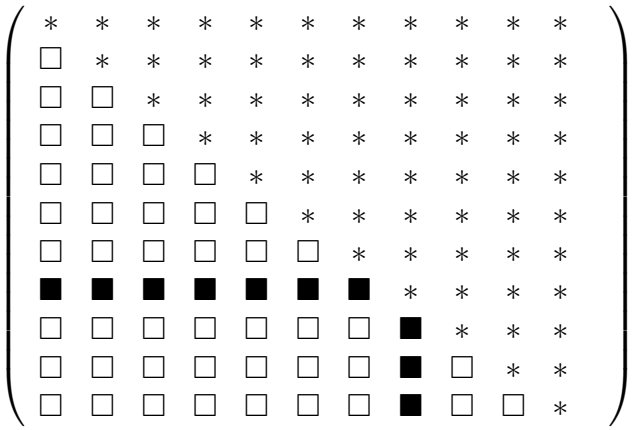
Der erste Fall ist angenehm, da wir nur einmal die Position $p(j)$ der Zeile für j bestimmen müssen, und danach für jedes $1 \leq i < j$ die Einträge $p(j)+i-1$ nacheinander ohne weitere Multiplikation zu erreichen sind.

Um nun ohne weitere Multiplikationen auch die Positionen $p(i) + j - 1$ für $i > j$ zu bestimmen, betrachten wir Differenz der zugehörigen Positionen für zwei aufeinanderfolgende i:

$$\Big(\frac{i(i-1)}{2} + j - 1\Big) - \Big(\frac{(i-1)(i-2)}{2} + j - 1\Big) = \frac{i(i-1) - (i-1)(i-2)}{2} = i - 1.$$

Wenn wir uns also derzeit bereits an Position $p(i) + j - 1$ im Array befinden, weil wir die Adjazenz von i und j für $i > j$ überprüft haben, dann erreichen wir den Eintrag an Position $p(i+1) + j - 1$ für die Adjazenz der Knoten $i+1$ und j, indem wir $i - 1$ zur aktuellen Position hinzuaddieren. Die einzige Ausnahme bildet Übergang von $i = j - 1$ auf $i = j+1$. Aufgrund des fehlenden Eintrags $i = j$ müssen wir in diesem Fall j Einträge weitergehen.

Wir können also, indem wir anfangs einmal zwei Multiplikationen verwenden, um die Position $p(j)$ zu bestimmen, danach unter Verwendung von Additionen sehr einfach durch die benötigen Einträge im Array zugreifen. Die entsprechende Technik nennt sich „lineare Fortschaltung“. Anschaulich wandern wir dann folgendermaßen durch die Dreiecksmatrix, wobei der Eintrag für $i = j$ übersprungen wird.



Da wir für jedes $1 \leq i \leq n$ nur eine Position besuchen müssen, die sich jeweils in konstant vielen Schritten aus der vorherigen Position bestimmen läßt, benötigt das Verfahren dann $\Theta(n)$ Schritte.

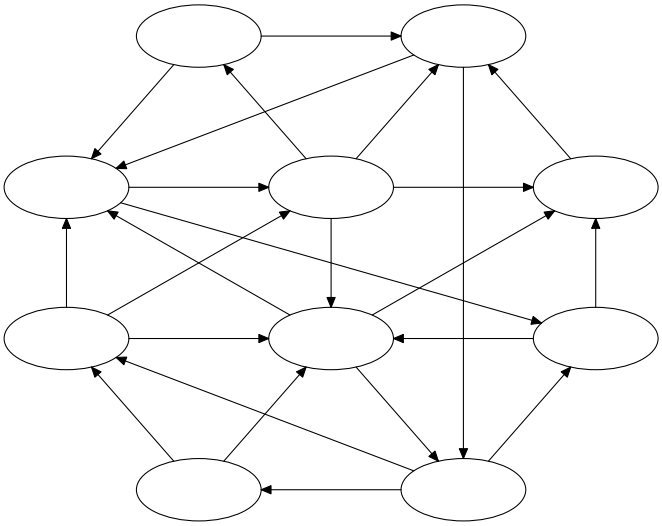
Wir können mit einem weiteren Trick sogar ganz ohne Multiplikationen auskommen: Da wir sowieso die Position $p(n) + j - 1$ überprüfen müssen, können wir anstatt von oben links nach rechts unten auch umgekehrt von rechts unten nach oben links wandern. Die Position $p(n) = (n-1)(n-2)/2$ ist unabhängig von j und ändert sich nicht, solange sich die Anzahl der Knoten n im Graphen nicht ändert. Wenn wir also zusätzlich in einer speziellen Variablen diese Position $p(n)$ speichern und nur dann updaten, wenn sich die Anzahl der Knoten im Graph ändert, läßt sich die Nachbarschaft eines Knotens sogar ganz ohne Multiplikationen bestimmen.

Aufgabe H21 (10 Punkte)

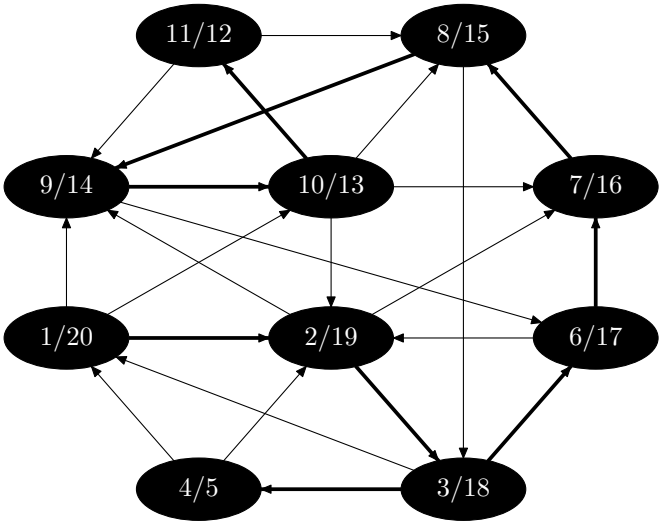
Beweisen oder widerlegen Sie die folgende Aussage: Wenn eine Tiefensuche auf einem ungerichteten Graphen durchgeführt wird, erhalten wir keine Querkanten.

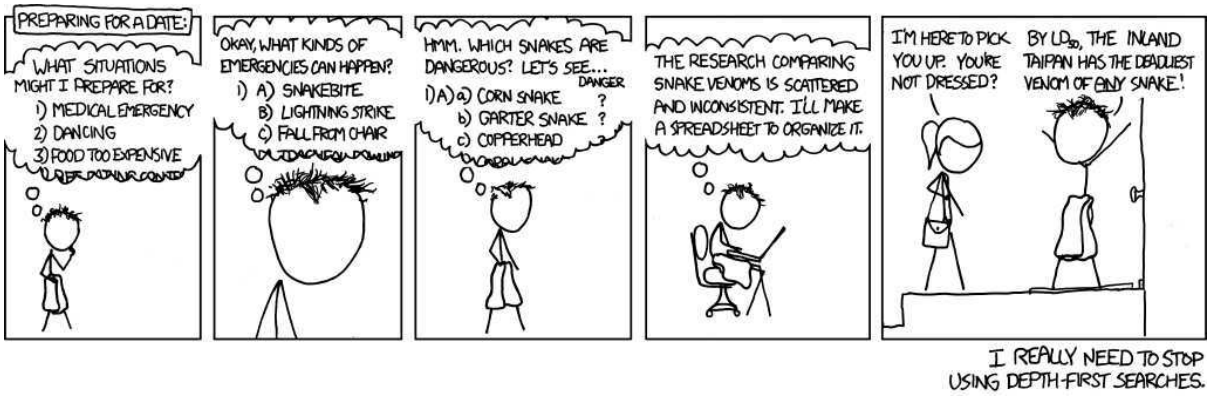
Aufgabe H22 (10 Punkte)

Führen Sie eine Tiefensuche auf dem folgenden Graphen durch. Geben Sie zu jedem Knoten *discovery* und *finish* Zeiten an, und geben Sie dann zu jeder Kante an, welchen Typ sie bezüglich des entstandenen Tiefensuchwaldes hat (Baumkante, Vorwärtskante, Rückwärtskante, Querkante).



Lösungsvorschlag:





(Erlaubter Abdruck von der Webseite `http://xkcd.com/`)

RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Übungsblatt 9
Emmes—Langer—Reidl—Rossmanith 06.06.2011

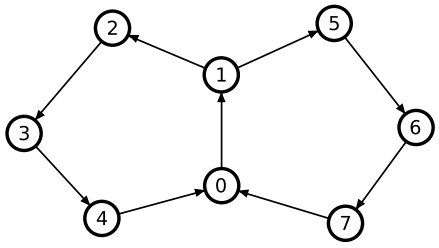
Übung zur Vorlesung Datenstrukturen und Algorithmen

Aufgabe T28

Beweisen oder widerlegen Sie: Ergibt eine Tiefensuche in einem gerichteten Graphen genau eine Rückwärtskante, so liefert jede Tiefensuche in diesem Graphen genau eine Rückwärtskante.

Lösungsvorschlag:

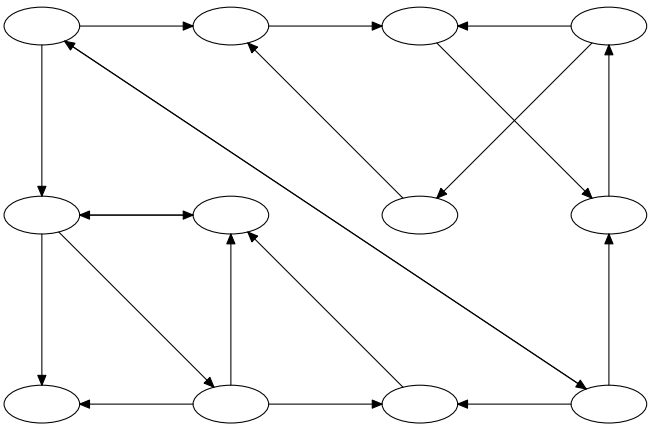
Die Aussage ist falsch, wie man sich an folgendem Gegenbeispiel überlegen kann.



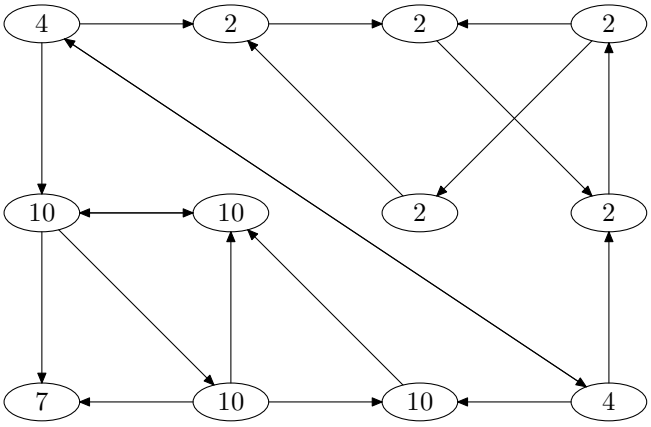
Beginnen wir die Tiefensuche am Knoten 1, so findet man genau eine Rückwärtskante, nämlich $(0, 1)$ (egal welchen Kreis man zuerst besucht). Eine Tiefensuche, die am Knoten 0 startet, wird jedoch die Rückwärtskanten $(4, 0)$ und $(7, 0)$ finden!

Aufgabe T29

Bestimmen Sie die starken Zusammenhangskomponenten des folgenden Graphen, indem Sie den Algorithmus von Kosaraju verwenden.



Lösungsvorschlag:



Aufgabe T30

Jemand behauptet, der Algorithmus zum topologischen Sortieren aus der Vorlesung funktioniert insofern auch auf Graphen, die Zyklen erhalten, als daß er Sortierungen mit der folgenden Eigenschaft liefert:

> Wenn zwei Knoten $u, v \in V$ in verschiedenen starken Zusammenhangskomponenten liegen, dann werden sie korrekt geordnet.

Beweisen Sie diese Aussage und überlegen Sie sich ein Anwendungsbeispiel, oder widerlegen Sie sie und entwerfen Sie einen Ersatzalgorithmus.

Lösungsvorschlag:

Die Aussage ist falsch. Dies sieht man leicht, indem man zwei gerichtete Kreise der Länge drei durch eine gerichtete Kante verbindet und die Suche an einem ungünstigen Knoten im ersten Kreis beginnt und dann zuerst in den zweiten Kreis hineinläuft, bevor man den ersten abarbeitet.

Der Ersatzalgorithmus kann zum Beispiel wie folgt aussehen.

1. Schrumpfe die starken Zusammenhangskomponenten jeweils zu einzelnen Knoten zusammen.
2. Sortiere diesen Komponentengraphen topologisch.
3. Expandiere die Komponenten in der ermittelten Reihenfolge. Hierbei spielt die Reihenfolge innerhalb jeder Komponente keine Rolle.

Offensichtlich leistet dieses Verfahren das Gewünschte.

Aufgabe H23 (15 Punkte)

Beweisen oder widerlegen Sie: Ergibt eine Tiefensuche in einem ungerichteten Graphen genau eine Rückwärtskante, so liefert jede Tiefensuche in diesem Graphen genau eine Rückwärtskante.

Lösungsvorschlag:

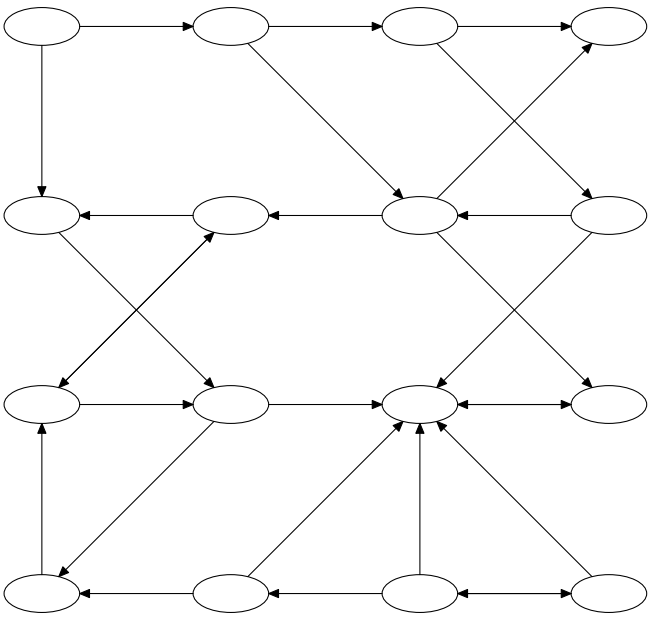
Stellen wir zunächst fest, daß wir uns auf zusammenängende Graphen beschränken können: sollte der Graph mehrere Komponenten besitzen, so kann natürlich nur in einer eine Rückwärtskante auftauchen.

Wir benutzen nun die Tatsache, daß Bäume mit n Knoten genau $n-1$ Kanten besitzen, also $n = m + 1$ gilt. Dies kann man sich einfach veranschaulichen, indem man in einem Baum willkürlich eine Wurzel festlegt und alle Kanten von der Wurzel weg orientiert—auf diese Weise zeigt nun genau eine Kante auf jeden Knoten, abgesehen von der Wurzel, auf welche keine Kante zeigt.

In unserem Tiefensuchbaum gilt nun $n = m$, denn neben den $n-1$ Baumkanten existiert noch die eine Rückwärtskante. Vor diesem Hintergrund ergibt sich die obige Aussage sehr einfach: jeder Tiefensuchbaum muß, da wir ihn als Spannbaum unseres Graphen auffassen können, $n-1$ Baumkanten besitzen. Da der Graph n Kanten besitzt, muß die verbleibende Kante eine Rückwärtskante sein (man beachte, daß dies in gerichteten Graphen nicht folgt: dort kann sie auch eine Querkante sein).

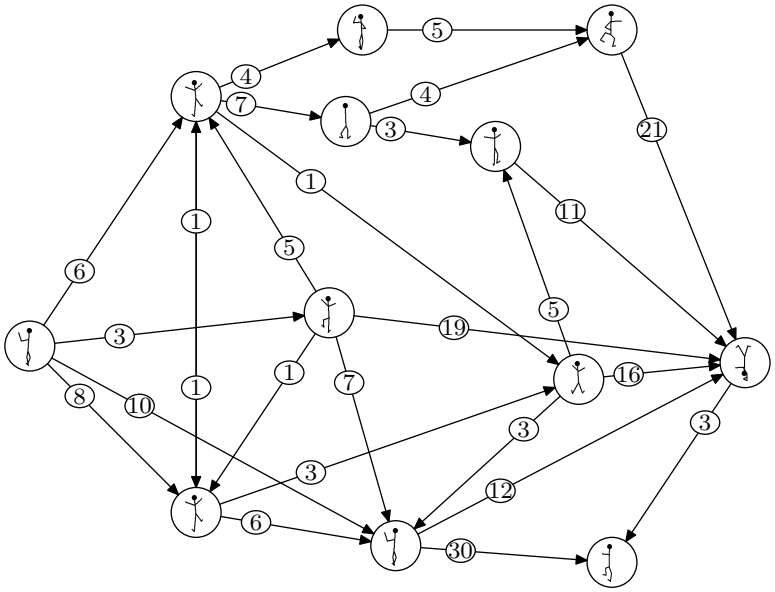
Aufgabe H24 (10 Punkte)

Bestimmen Sie die starken Zusammenhangskomponenten des folgenden Graphen, indem Sie den Algorithmus von Kosaraju verwenden.

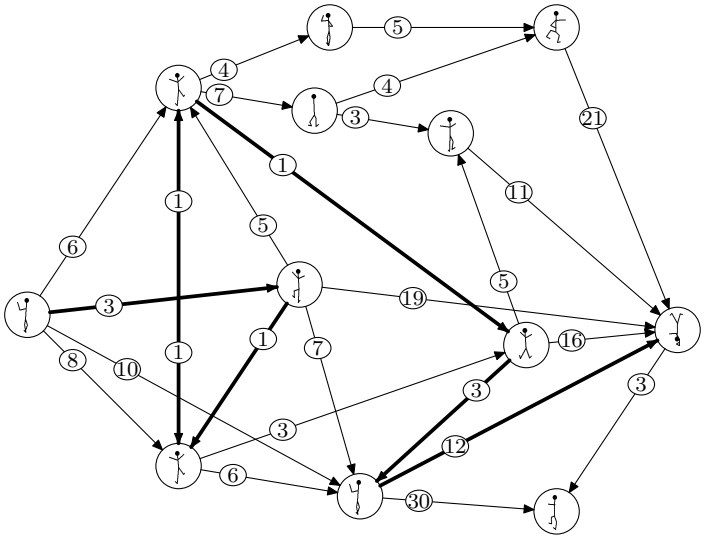


Aufgabe H25 (10 Punkte)

Berechnen Sie mit Hilfe des Algorithmus von Dijkstra den kürzesten Pfad vom linken Knoten zum rechten Knoten.



Lösungsvorschlag:
Der eingezeichnete Pfad ist korrekt:



RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Übungsblatt 10
Emmes—Langer—Reidl—Rossmanith 20.06.2011

Übung zur Vorlesung Datenstrukturen und Algorithmen

Aufgabe T31

Betrachten Sie den verbesserten Algorithmus aus der Vorlesung, um die transitive Hülle eines gerichteten Graphens $G = (V, E)$ mit $|V| = n$ Knoten zu berechnen. Wie lange benötigt dieser Algorithmus auf den folgenden Graphfamilien? Besteht eine Verbesserung gegenüber der direkten Anwendung des Algorithmus von Warshall?

- DAGs
- Hamiltonische Graphen, in denen es einen Kreis der Länge n gibt.
- Dreiecksgraphen, die aus $n/3$ disjunkten, nicht verbundenen Dreiecken bestehen.
- Graphen, die aus $\sqrt{n}$ vielen disjunkten, nicht verbundenen Kreisen der Länge jeweils $\sqrt{n}$ bestehen.

Lösungsvorschlag:

Der verbesserte Algorithmus aus der Vorlesung benötigt $O(|V|+|E'|+k^3)$ Schritte, wenn k die Anzahl der starken Zusammenhangskomponenten und E' die Kanten der transitiven Hülle sind. Der Algorithmus von Warshall benötigt $O(n^3)$ Schritte. Das Berechnen der starken Zusammenhangskomponenten benötigt mit dem Algorithmus von Kosaraju jeweils lineare Zeit.

In DAGs ist jeder Knoten eine eigenständige starke Zusammenhangskomponente, so daß wegen $k = n$ auch Laufzeit $O(n^3)$ erreicht wird.

In Hamiltonischen Graphen gibt es nur eine starke Zusammenhangskomponente. Die transitive Hülle auf dem resultierenden geschrumpften Graphen mit einem Knoten läßt sich dann sogar in konstanter Zeit berechnen. Jedoch enthält E' dann n^2 viele Kanten, so daß die Laufzeit insgesamt $O(n^2)$ beträgt.

In Dreiecksgraphen ist jedes Dreieck eine starke Zusammenhangskomponente. Jedes dieser $k = n/3$ Dreiecke wird zu einem Knoten geschrumpft, so daß im Algorithmus die transitive Hülle eines Graphens mit $k = n/3$ Knoten berechnet wird. Dieses benötigt also auch $O(n^3)$ Schritte, welches die Gesamtlaufzeit dominiert.

In einem Graphen mit $\sqrt{n}$ Kreisen der Länge jeweils $\sqrt{n}$ ist jeder Kreis eine starke Zusammenhangskomponente, also $k = \sqrt{n}$. Es wird also die transitive Hülle eines Graphens mit k Knoten berechnet, welches $O(n^{3/2})$ Schritte benötigt. In E' gibt es dann nur Kanten innerhalb der starken Zusammenhangskomponenten, dort aber von jedem Konten zu jedem Knoten der starken Zusammenhangskomponente, so daß es in E' insgesamt n Kanten pro Zusammenhangskomponente gibt und $|E'| = n\sqrt{n}$ gilt. Die Gesamtlaufzeit beträgt also nur $O(n\sqrt{n} + n^{3/2}) = O(n^{3/2})$.

Aufgabe T32

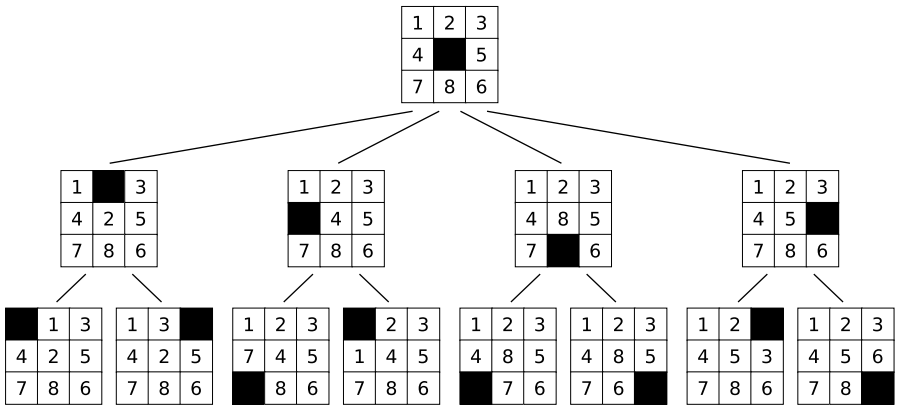
Wir sagen, daß ein Schiebepuzzle der Dimension $m \times m$ gelöst ist, wenn das leere Feld (hier abgebildet in schwarz) unten rechts an Position (m, m) liegt und die anderen Steine korrekt in aufsteigender Reihenfolge (in Leserichtung) vorliegen. Gegeben sei nun das folgende 3×3-Schiebepuzzle. Finden Sie mit Hilfe einer Breitensuche eine minimale Folge von Verschiebeoperationen, welche das Puzzle löst.

1	2	3
4		5
7	8	6

Lösungsvorschlag:

Wir konstruieren einen Konfigurationsgraphen mit Hilfe einer Breitensuche wie folgt: Ausgehend von der Eingabekonfiguration wenden wir jede mögliche Verschiebeoperation an und erhalten so eine neue Konfiguration. Die beiden Konfigurationen verbinden wir über eine Kante, da jede Konfiguration aus der jeweils anderen erreichbar ist. Bereits betrachtete Konfigurationen müssen nicht erneut besucht werden. Zusätzlich läßt sich wegen der Breitensuche so auch eine kürzeste Verschiebesequenz finden, da lange Sequenzen erst dann betrachtet werden, wenn alle kürzeren Sequenzen bereits abgearbeitet wurden.

In der folgenden Lösung für obiges Puzzle haben wir mit der gegebenen Initialkonfiguration ganz oben begonnen, aus der sich die Konfigurationen in der zweiten Zeile ergeben haben, die in der abgebildeten Reihenfolge (von links nach rechts) in die *queue* eingefügt wurden. Für jede der Konfigurationen der zweiten Zeile haben wir dann die Nachfolgekonfigurationen berechnet, aber natürlich bereits besuchte Konfigurationen (hier: die Initialkonfiguration) nicht erneut besucht. Die gesucht Verschiebesequenz $5, 6$, welches das Puzzle löst, ergab sich in diesem Fall ganz am Ende.



Aufgabe T33

Entwerfen Sie einen Algorithmus, der in einem gerichteten Graphen für zwei Knoten s, t alle Pfade von s nach t ausgibt. Welche Laufzeit hat dieser Algorithmus in Bezug auf die Zahl der ausgegebenen Pfade? Wie läßt sich der Algorithmus für Graphen beschleunigen, in denen es viele Knoten gibt, die von s aus erreichbar sind, von denen allerdings kein Pfad zu t existiert?

Lösungsvorschlag:

In folgendem Algorithmus enthält das Array v bis zum Index k alle Knoten des Graphen, die bisher besucht wurden. Die Suche wird dann mit den direkten Nachfolgern von $v[k]$ fortgeführt, die noch nicht besucht wurden. Die Funktion N gibt hier die direkten Nachfolger eines Knoten zurück. Der Test ob u nicht in v liegt, läßt sich durch geeignete Markierung des Graphen in konstanter Zeit realisieren. Beim Erreichen des Knoten t, wird der Pfad ausgegeben.

```
procedure enumeratePaths(s, t):
v[0] = s;
enum(s, v, 0, t);

procedure enum(s, v, k, t):
if v[k] = t then
    print(v[0],..., v[k]);
else
    forall u in N(v[k]) do
        if u not in v then
            v[k+1] ← u;
            enum(s, v, k+1, t);
        fi
    od
fi
```

Offensichtlich werden alle möglichen Pfade von s nach t durch diesen Algorithmus aufgezählt. In Bezug auf die Zahl der existierenden Pfade von s nach t ist dies allerdings nicht effizient, wie folgendes Beispiel beleuchtet:

Sei der Knoten s eines Graphen mit Knotenzahl n in einem vollständigen Teilgraphen der Größe $n-1$ enthalten und habe der Knoten t keine eingehenden Kanten. Dann durchläuft der Algorithmus alle $(n-1)!$ Pfade in dem vollständigen Teilgraphen, ohne einen Pfad zu t zu finden.

Um dieses Verhalten zu verbessern, beschränkt man die Suche auf die Knoten, von denen aus der Endknoten t erreichbar ist. Dieses läßt sich ganz zu Beginn durch eine Tiefensuche feststellen. Da sich während der Berechnung des Algorithmus die Menge der erlaubten Knoten ändert (schon besuchte Knoten dürfen nicht wieder besucht werden), reicht es nicht, nur anfänglich die Knoten zu bestimmen, von denen aus t erreichbar ist. Deswegen testet man dies vor dem rekursiven Aufruf der Prozedur erneut.

```
procedure enum(s, v, k, t):
if v[k] = t then
    print(v[0],..., v[k]);
else
    forall u in N(v[k]) do
        if u not in v ∧ t reachable from u without using nodes in v then
            v[k+1] ← u;
            enum(s, v, k+1, t);
        fi
    od
fi
```

Nun ist sichergestellt, dass jeder rekursive Aufruf mindestens zu einem ausgegebenen Pfad nach t führt. Da die maximale Pfadlänge durch die Größe des Graphen n beschränkt ist, haben wir pro ausgegebenen Pfad höchstens n Überprüfungen auf Erreichbarkeit. Diese lassen sich jeweils in linearer Zeit im Verhältnis zur Graphgröße n durchführen. Insgesamt ergibt sich also eine Zeitkomplexität von $O(n^2)$ pro ausgegebenem Pfad.

Aufgabe H26 (15 Punkte)

Implementieren Sie in einer Programmiersprache Ihrer Wahl ein Programm, das für ein beliebiges Schiebepuzzle der Dimension 4×4 eine möglichst kurze Folge von Verschiebeoperationen ausgibt, mit der sich das Puzzle lösen läßt, falls eine solche Folge existiert. Ein Beispiel ist unten abgebildet.

Auf der Homepage zur Vorlesung finden Sie zehn Textdateien mit Schiebepuzzeln der Dimension 4×4. Geben Sie für jedes dieser Puzzle, falls lösbar, eine möglichst kurze Folge von Verschiebeoperationen an, die das jeweilige Puzzle löst.

Bitte geben Sie auch die Zeit an, die Ihr Programm benötigt hat, um die jeweiligen Instanzen zu lösen, und wie viele Konfigurationen Ihr Programm dabei untersucht hat. Die Laufzeit und die Anzahl besuchter Konfigurationen hat keinen Einfluß auf die Bewertung.

1	3	6	4
5		7	8
9	2	F	B
D	A	E	C

Lösungsvorschlag:

Ein Beispielprogramm, welches Breitensuche verwendet, findet sich auf der Homepage zur Vorlesung.

Aufgabe H27 (5 Punkte)

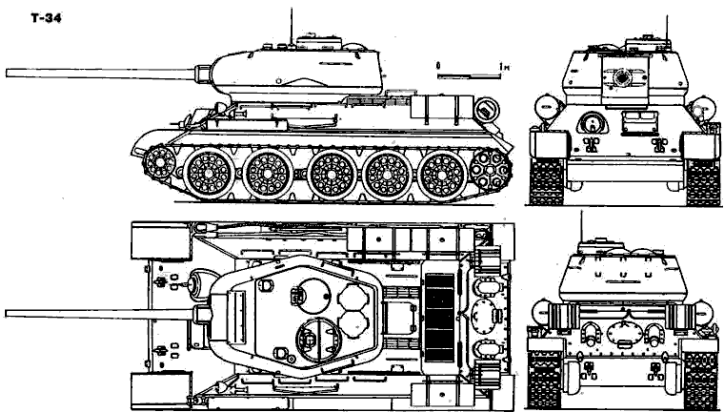
Konstruieren Sie einen Algorithmus, der auf einem gewichteten, gerichteten Graphen einen kürzesten Pfad zwischen zwei Knoten s und t berechnet. Der Graph darf negative Kantengewichte und Kreise mit negativem Gesamtgewicht enthalten.

Lösungsvorschlag:

Da wir keine Anforderungen an die Laufzeit des Algorithmus haben, begnügen wir uns damit, die Liste aller Pfade von s nach t mit Hilfe des Algorithmus aus T33 zu generieren und sie anschließend nach dem kürzesten zu durchsuchen.

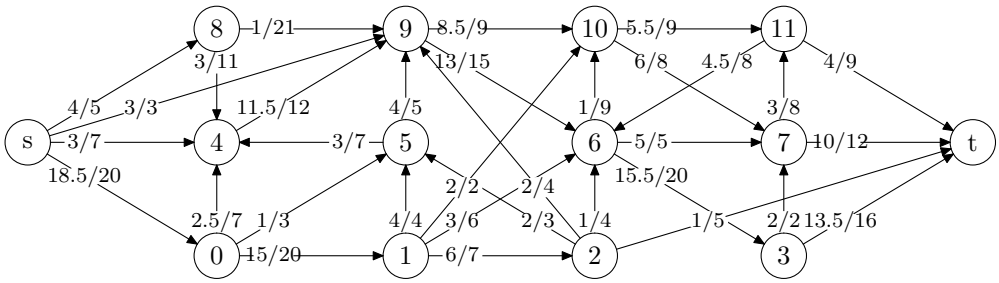
RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Übungsblatt 11
Emmes—Langer—Reidl—Rossmanith 27.06.2011

Übung zur Vorlesung Datenstrukturen und Algorithmen



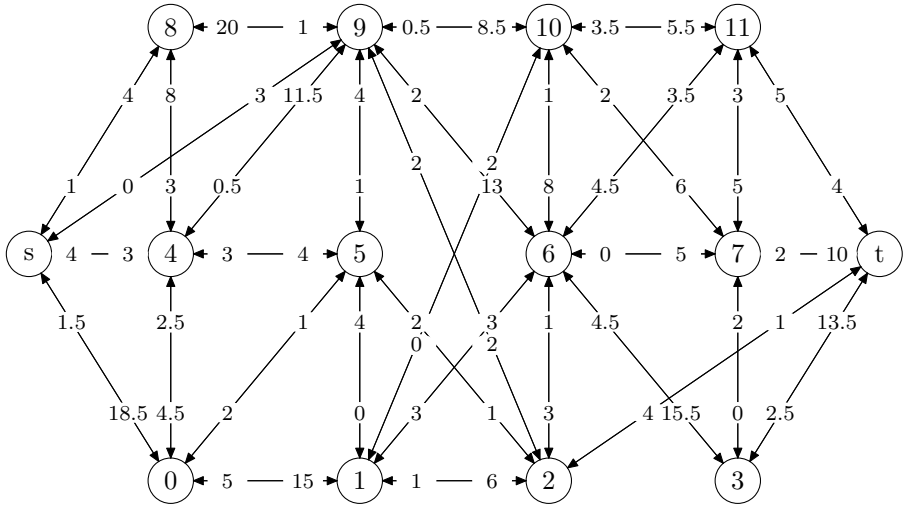
T35

Bestimmen Sie das Residualnetzwerk des folgenden Netzwerkes mit angegebenem Fluß f.



Lösungsvorschlag:

Es ergibt sich das folgende Residualnetzwerk. Die Kapazität einer Kante ist jeweils auf 1/3 der Strecke eingezeichnet, welche die Kante aufspannt.



T36

Beweisen Sie den zweiten Punkt von Lemma A: $f(X,Y) = -f(Y,X)$ für $X, Y \subseteq V$ falls f ein Fluß für $G = (V, E)$ ist.

Lösungsvorschlag:

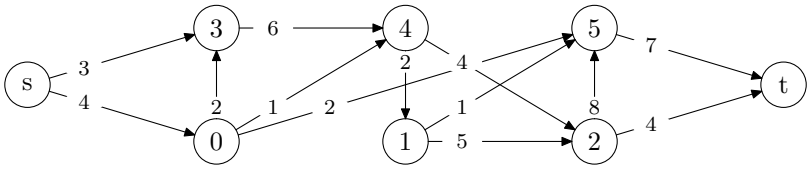
$$f(X,Y) \overset{Def.}{=} \sum_{x \in X} \sum_{y \in Y} f(x,y) \overset{Symm.}{=} \sum_{x \in X} \sum_{y \in Y} -f(y,x) =$$

$$\sum_{y \in Y} \sum_{x \in X} -f(y,x) = -\sum_{y \in Y} \sum_{x \in X} f(y,x) \overset{Def.}{=} -f(Y,X)$$

Aufgepasst: Welche Rolle spielen hier das Kommutativitäts-, das Assoziativ- und das Distributivgesetz?

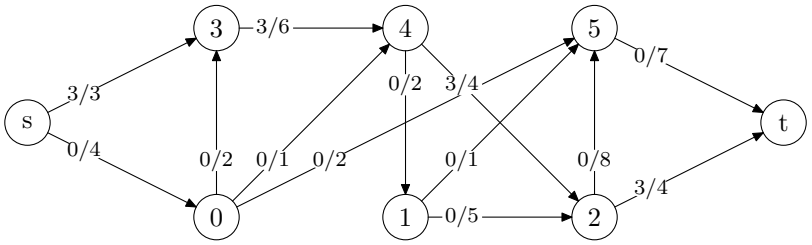
T37

Bestimmen Sie den maximalen Fluß des folgenden Netzwerkes mit Hilfe der Methode von Ford und Fulkerson.

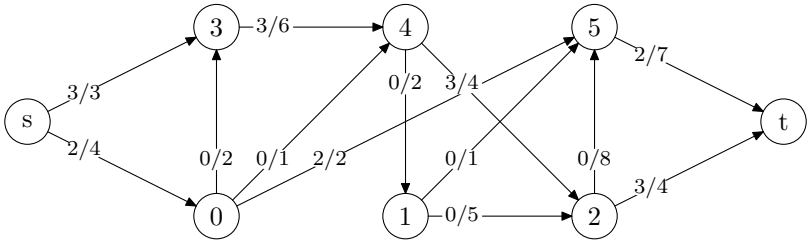


Lösungsvorschlag:

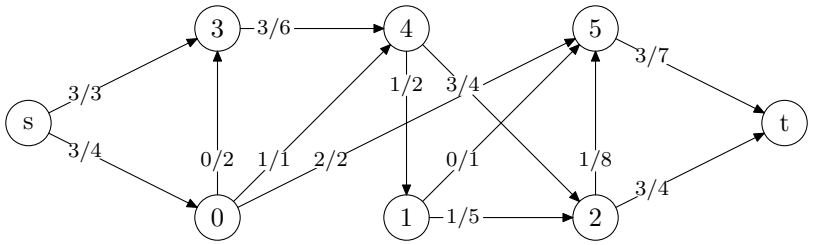
Augmentiere Pfad $s, 3, 4, 2, t$:



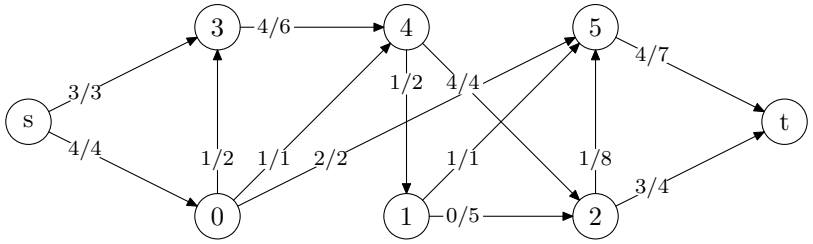
Augmentiere Pfad $s, 0, 5, t$:



Augmentiere Pfad $s, 0, 4, 1, 2, 5, t$:



Augmentiere Pfad $s, 0, 3, 4, 2, 1, 5, t$:



Der maximale Fluß beträgt also 7.

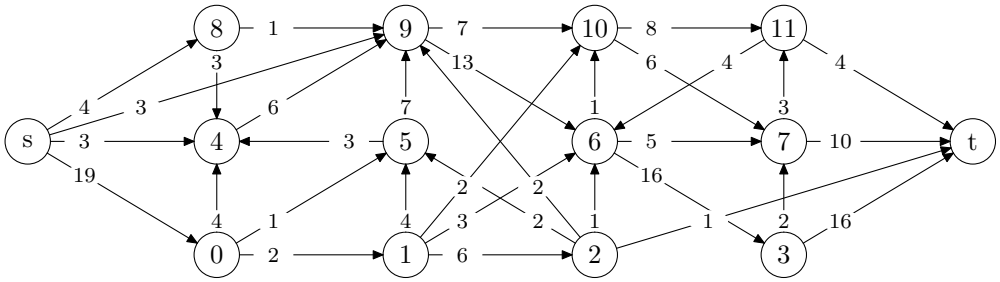
H28 (5 Punkte)
Beweisen Sie den dritten Punkt von Lemma A: $f(X \cup Y, Z) = f(X, Z) + f(Y, Z)$ für $X, Y, Z \subseteq V$, $X \cap Y = \emptyset$, falls f ein Fluß für $G = (V, E)$ ist.

Lösungsvorschlag:

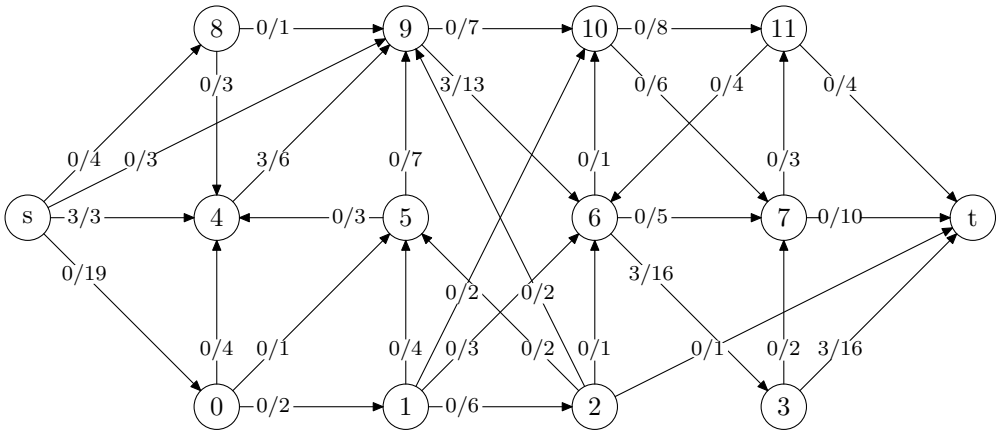
$$f(X \dot\cup Y, Z) \overset{Def.}{=} \sum_{v \in X \dot\cup Y} \sum_{z \in Z} f(v, z) =$$

$$\sum_{x \in X} \sum_{z \in Z} f(x, z) + \sum_{y \in Y} \sum_{z \in Z} f(y, z) \overset{Def.}{=} f(X, Z) + f(Y, Z)$$

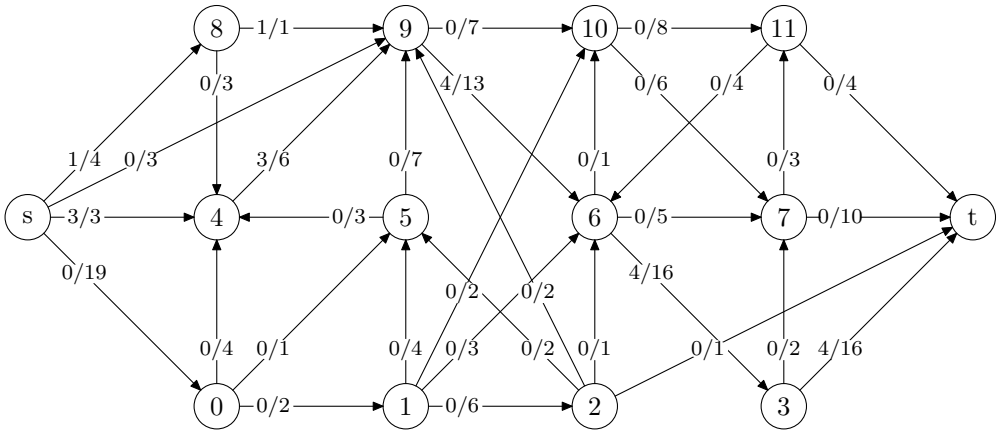
H29 (10 Punkte)
Bestimmen Sie den maximalen Fluß des folgenden Netzwerkes mit Hilfe der Methode von Ford und Fulkerson.



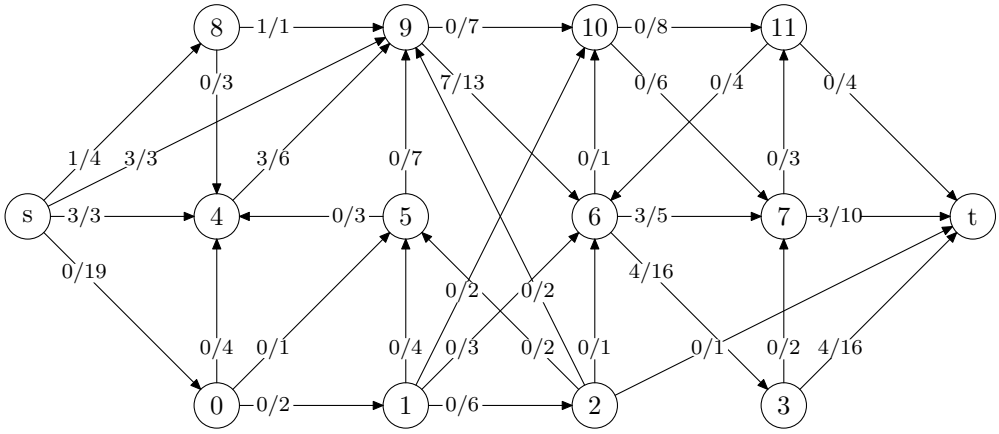
Lösungsvorschlag:
Augmentiere Pfad $s, 4, 9, 6, 3, t$:



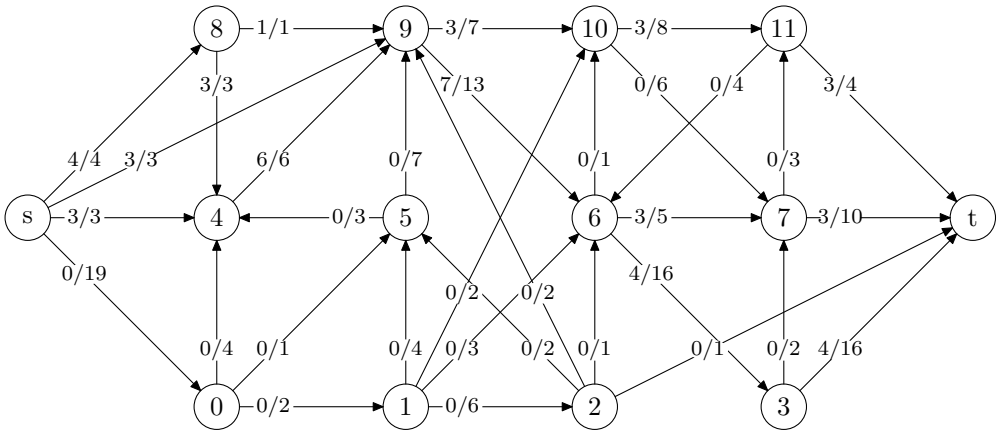
Augmentiere Pfad $s, 8, 9, 6, 3, t$:



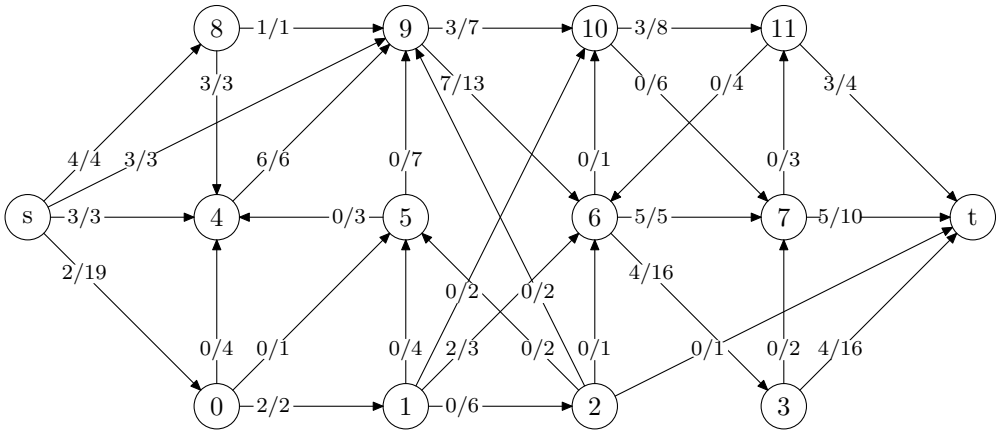
Augmentiere Pfad $s, 9, 6, 7, t$:



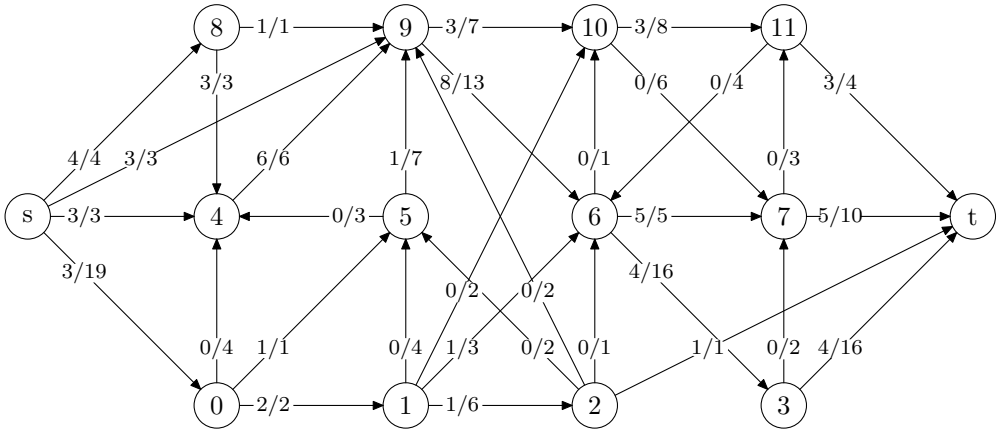
Augmentiere Pfad $s, 8, 4, 9, 10, 11, t$:



Augmentiere Pfad $s, 0, 1, 6, 7, t$:



Augmentiere Pfad $s, 0, 5, 9, 6, 1, 2, t$:



Der maximale Fluß beträgt also 13.

H30 (10 Punkte)

Der niederländische Fritten-Franchise 'Vet Druipend' hat die Stadt Heinsberg erschlossen, dabei jedoch den Markt übersättigt: an vielen Straßenkreuzungen stehen schon mehrere Pommesmobile, in einigen Straßen macht sich 'Vet Druipend' also schon selbst Konkurrenz! Das soll sich nun ändern: künftig werden anstatt Kreuzungen einzelne Straßen von jeweils nur noch einem Mobil bedient. Aus Kostengründen soll diese Umstrukturierung

geschehen, indem Mobile von den Kreuzungen in anliegende Straßen geschoben werden—natürlich soll weiterhin ganz Heinsberg bedient werden, es muß also jede Straße abgedeckt werden (Mobile, die bei dieser Maßnahme übrigbleiben, werden von 'Vet Druipend' abgestoßen).
Vereinfachen wir das Problem zu einem Graphenproblem. Sei G ein Graph mit einer Knotenbeschriftung $p\colon V(G) \to \mathbf{N}$. Jeder Knoten $v \in V(G)$ kann bis zu $p(v)$ benachbarte Kanten abdecken. Gefragt ist, ob alle Kanten des Graphen so abgedeckt werden können. Formal suchen wir eine Funktion $g\colon E(G) \to V(G)$ mit den folgenden Eigenschaften:

1. $g(e) = v \Rightarrow e$ ist inzident zu v
2. $|g^{-1}(v)| \leq p(v)$ für alle $v \in V(G)$

Beschreiben Sie, wie das obige Problem als ganzzahliges Flußproblem modelliert werden kann. Genauer soll der maximale, ganzzahlige Fluß betraglich genau dann der Anzahl der Kanten entsprechen, falls eine solche Funktion g existiert.

Lösungsvorschlag:
Wir bauen folgendes s,t-Netzwerk N aus G:

1. N enthalte zunächst alle Knoten $V(G)$
2. Wir ergänzen die Quelle s und fügen gerichtete Kanten (s, v) mit Kapazität $p(v)$ für alle $v \in V(G)$ ein
3. Für jede Kante $e = (u, v) \in E(G)$ fügen wir einen neuen Knoten w_e ein, anschließend ergänzen wir die Kanten (u, w_e) und (v, w_e) mit Kapazität eins
4. Zuletzt führen wir eine Senke t ein und verbinden alle vorher eingefügten Knoten w_e mit t mittels Kanten (w_e, t) mit Kapazitkät eins

Es bleibt zu zeigen, daß N genau dann einen maximalen Fluß f mit $|f| = m$ hat, wenn für G eine wie oben beschriebenen Zuordnung g existiert.
Nehmen wir an, f sei ein maximaler Fluß mit Betrag m. Aufgrund der Flußerhaltung kann für jedes Paar u, v mit $e := (u, v) \in E(G)$ jeweils nur eine der Kanten (u, w_e), (v, w_e) in N durch f gesättigt sein, weiterhin fließen in jeden Knoten $v \in V(N) \cap V(G)$ maximal $p(v)$ Flußeinheiten. Damit erhalten wir eine gültige Zuordnung g wie folgt:

$$g(e) = \begin{cases} u & f \text{ sättigt } (u, w_e) \\ v & f \text{ sättigt } (v, w_e) \end{cases}$$

Nehmen wir umgekehrt an, G habe eine gültige Zuordnung g. Dann können wir leicht einen Fluß f für N konstruieren:

$$\begin{aligned} f(s, v) &= p(v) \text{ für } v \in V(G) && (1) \\ f(v, w_{(u,v)}) = 1 &\Leftrightarrow g((u, v)) = v && (2) \\ f(w_e, t) &= 1 && (3) \end{aligned}$$

Man überzeuge sich davon, daß f tatsächlich ein Fluß in N ergibt. Die Maximalität folgt schlicht aus der Tatsache, daß alle Kanten zu t saturiert sind.

RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Übungsblatt 12
Emmes—Langer—Reidl—Rossmanith 04.07.2011

Übung zur Vorlesung Datenstrukturen und Algorithmen

T38

Die internationale Raumstation ISS steht auch (zahlungswilligen) Weltraumtouristen offen. Sie sollen nun entscheiden, welche Touristen Sie mitnehmen wollen.

Gegeben sind Kandidaten $K_1, \ldots, K_n$, welche jeweils bereit sind, $k_1, \ldots, k_n$ US-Dollar zu zahlen. Allerdings sind sie anspruchsvoll und erwarten auf der ISS auch ein Unterhaltungsprogramm. Zu diesem Zweck stehen eine Menge „Spielzeuge“ $Z_1, \ldots, Z_m$ zur Verfügung, bei deren Mitnahme allerdings jeweils Kosten $z_1, \ldots, z_m$ entstehen. Der Kandidat K_i ist nur bereit zu zahlen, wenn die Spielzeuge $R_i \subseteq \{Z_1, \ldots, Z_m\}$ mitgenommen werden.

Entwerfen Sie einen effizienten Algorithmus, der eine Menge von Kandidaten auswählt, um die Einnahmen zu maximieren. Jedes Spielzeug muß nur einmal mitgenommen werden, selbst wenn mehrere es benutzen wollen.

Lösungsvorschlag:

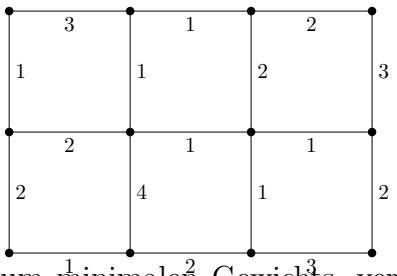
Wir konstruieren auf der Knotenmenge $\{s, t\} \cup \{K_1, \ldots, K_n\} \cup \{Z_1, \ldots, Z_m\}$ das folgende Flußproblem:

1. Von der Quelle s läuft zu jedem Knoten K_i eine Kante mit Kapazität k_i.
2. Für jedes $Z_j \in R_i$ läuft eine Kante mit unendlicher Kapazität von K_i nach Z_j.
3. Von jedem Knoten Z_i läuft eine Kante mit Kapazität z_i in die Senke t.

Ein Min-Cut für dieses Netzwerk ist natürlich endlich, da es endliche Schnitte gibt (beispielsweise die Menge aller Kanten, die in s beginnen). Die gestrichenen Kanten entsprechen jeweils verzichtetem Gewinn (durch Verzicht auf einen Touristen oder Verzicht auf die Ersparnis durch Nichtmitnahme eines Spielzeugs). Die Kanten mit unendlicher Kapazität können nicht saturiert werden und werden damit nie vom Schnitt betroffen sein. Somit entspricht jeder Min-Cut einer legalen Konfiguration (und andersherum).

T39

Führen Sie die Algorithmen von Prim auf folgendem Graphen aus.



Lösungsvorschlag:

Bei korrekter Ausführung ergibt sich jeweils ein Spannbaum minimalen Gewichts, vermutlich 15.

T40

Ein Gemischtwarenladen will durch einen besonderen
Rabatt die Umsätze steigern: Wer an der Kasse zwei Artikel zum Kauf vorlegt, deren Gesamtpreis auf 11, 33, 55, 77 oder 99 Cent endet, erhält zusätzlich einen Gutschein in Höhe des erreichten Centbetrages.

Entwerfen Sei einen Algorithmus, der zu einer Menge von Preisen eine optimale Einkaufsstrategie angibt.

Lösung:

Seien $X = \{x_1, \ldots, x_n\}$ die Preise der Gegenstände. Das Problem läßt sich leicht als Matchingproblem darstellen. Wir konstruieren einen bipartiten Graphen G mit den folgenden Knoten $G = \{x_i \in X \mid x_i \text{ ist gerade}\}$. Wir verbinden $x_i \in U$ mit $x_j \in G$, falls der Centbetrag c von $x_i + x_j$ entweder in 11, 33, 55, 77 oder in 99 ist. Die Kapazität der entsprechende Kante setzen wir auf c.

Es ist leicht einzusehen, daß ein maximales gewichtetes Matching in G genau eine Lösung unseres Problems darstellt. Formal wäre zu zeigen, daß jedes Matching eine Lösung unseres Problems darstellt und anders herum. Auf Grund der Konstruktion ist dies jedoch offensichtlich.

H31 (10 Punkte)

Die Produktionsaufträge $J_1, J_2, \ldots, J_n$ eines Unternehmens benötigen verschiedene Maschinen $M_1, M_2, \ldots, M_m$, wobei ein Auftrag auch mehrere Maschinen belegen kann. Ein Auftrag bringt natürlich einen gewissen Geldbetrag ein. Die Maschinen können entweder gekauft werden—diese Kosten entstehen dann nur einmal und jede weitere Nutzung ist Kostenfrei—oder gemietet werden. Letzteres kostet pro Auftrag einen gewissen Betrag (dieser Betrag variiert also von Auftrag zu Auftrag!).

Für so ein Szenario mit gegebenen Aufträgen, Maschinen und Kosten soll eine gewinnmaximierende Menge von Aufträgen mit entsprechender Zuteilung von Maschinen berechnet werden. Beschreiben Sie ein allgemeines Verfahren, um dieses Problem möglichst effizient zu lösen. Begründen Sie, wieso Ihr Verfahren korrekt funktioniert.

Benutzen Sie Ihr Verfahren, um eine optimale Lösung für das folgende Szenario zu berechnen. Die dritte Tabelle enthält die Mietkosten der Maschinen für die jeweiligen Aufträge, eine leere Zelle bedeutet, daß diese Maschine für diesen Auftrag nicht benötigt wird. Zum Beispiel kostet Auftrag J_1 auf der Maschine M_1 30 Geldeinheiten (vorausgesetzt, M_1 wurde nicht gekauft).

Auftrag	Zahlung
J_1	80
J_2	80
J_3	120

Maschine	Kaufpreis
M_1	60
M_2	80
M_3	100
M_4	30

	M_1	M_2	M_3	M_4
J_1	30		50	
J_2		60	22	
J_3	30	30	30	30

Lösungsvorschlag:

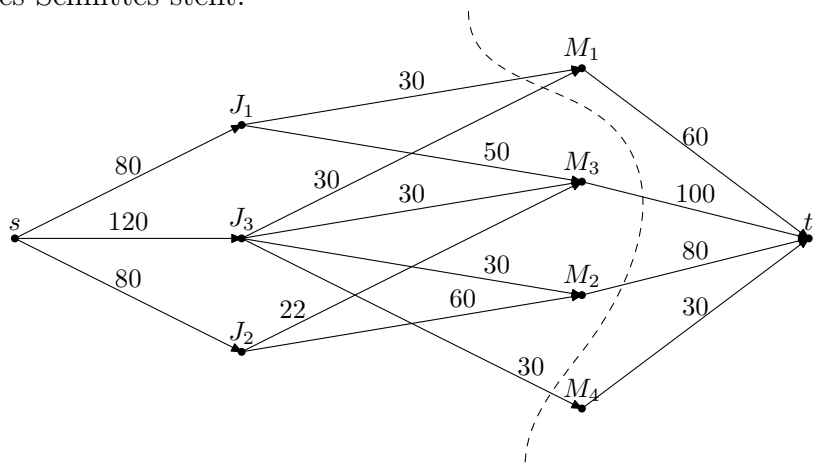
Wir verwenden eine Konstruktion analog zur Aufgabe T38: Die Knotenmenge des Flußnetzwerks ist $\{s, t\} \cup \{J_1, \ldots, J_n\} \cup \{M_1, \ldots, M_m\}$. Die Kanten des Netzwerks sind wie folgt:

1. Von der Quelle s läuft zu jedem Knoten J_i eine Kante mit Kapazität j_i, wenn j_i die für einen angenommenen Auftrag J_i erhaltene Zahlung ist.

2. Von einem Auftrag J_i läuft eine Kante zur Maschine M_j, wenn M_j für den Auftrag J_i benötigt wird. Die Kapazität auf dieser Kante sind die Kosten, die für eine einmalige Anmietung der Maschine M_j für den Auftrag J_i entstehen (Tabelle 3).

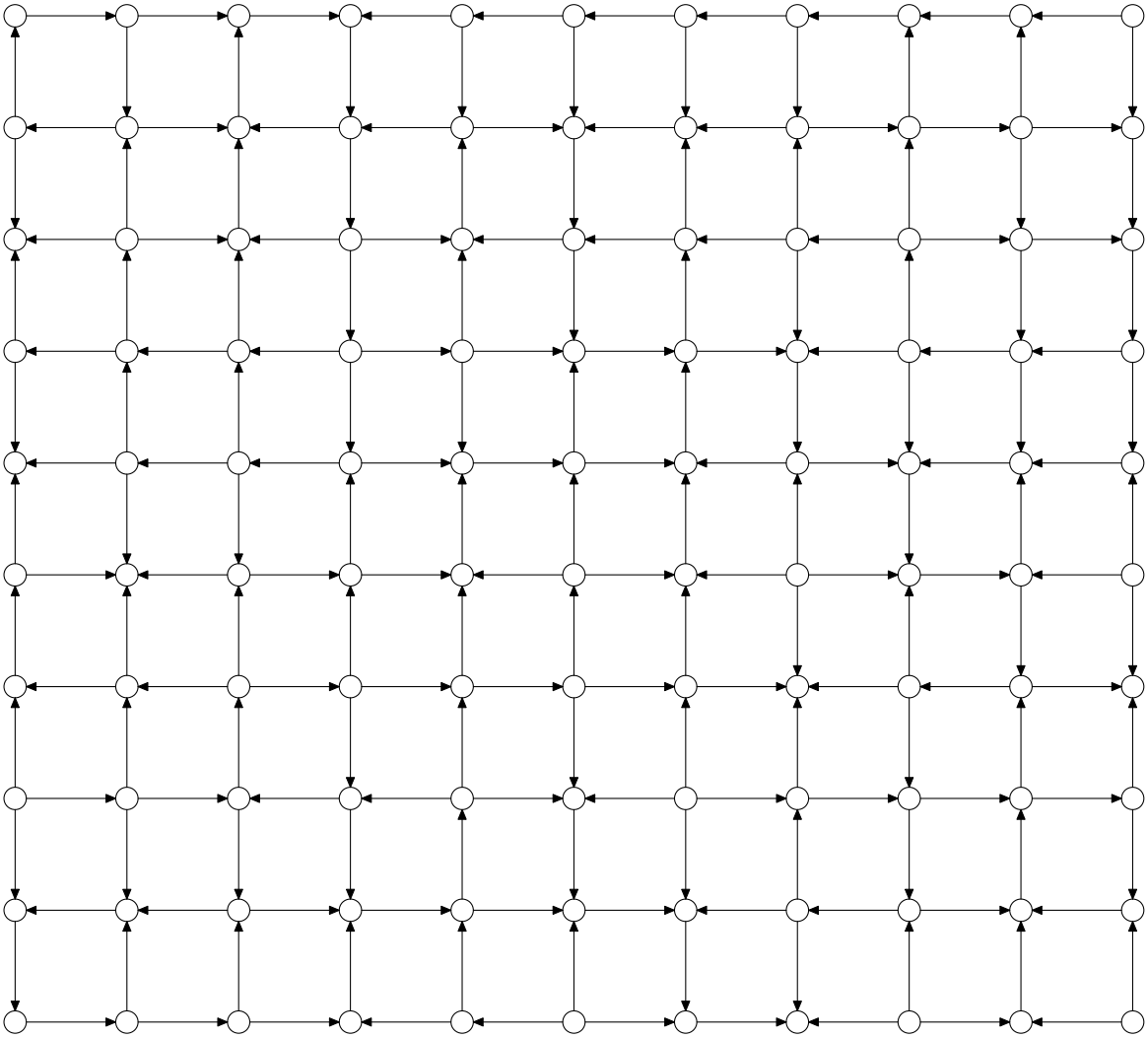
3. Von jedem Knoten M_i läuft eine Kante mit Kapazität m_i in die Senke t, wenn m_i die Kosten sind, um die Maschine M_i zu kaufen.

Ein Min-Cut (S, T) für dieses Netzwerk ist natürlich endlich und ganzzahlig und kann effizient mit der Methode von Ford und Fulkerson berechnet werden. Wir wählen nun diejenigen Aufträge aus, die in S enthalten sind. Ebenso kaufen wir alle Maschinen in S. Zur Begründung, warum dieses Verfahren korrekt ist: Die Kanten im Schnitt entsprechen wie in Aufgabe T38 anschaulich einem *Verzicht* auf einen Geldbetrag, welche der Kapazität dieser Kante entspricht. Wird etwa eine Kante $s \to J_i$ für einen Auftrag J_i vom Schnitt (S, T) geschnitten, dann verzichtet man auf die Einnahmen in Höhe von j_i. Der Verzicht auf die Einnahmen kann günstiger sein (und zu einem kleineren Schnitt führen), als wenn man den Job annimmt und dafür viele andere Kanten (für Aufträge oder Maschinen) schneiden muß. Schneidet man analog eine Kante $M_i \to t$, dann muß man den entsprechenden Kaufpreis für eine Maschine bezahlen. Auch dieses kann günstiger sein, als die Maschine für jeden angenommen Job zu mieten. Liegt andererseits eine Maschine M_j in T, dann ist es offenbar günstiger, M_j für jeden angenommenen Auftrag J_i anzumieten; andernfalls könnte man im leicht einen kleineren Schnitt konstruieren, indem man M_j nach S verschiebt (und dem Fall entspricht, M_j zu kaufen), was im Widerspruch zur Minimalität des Schnittes steht.

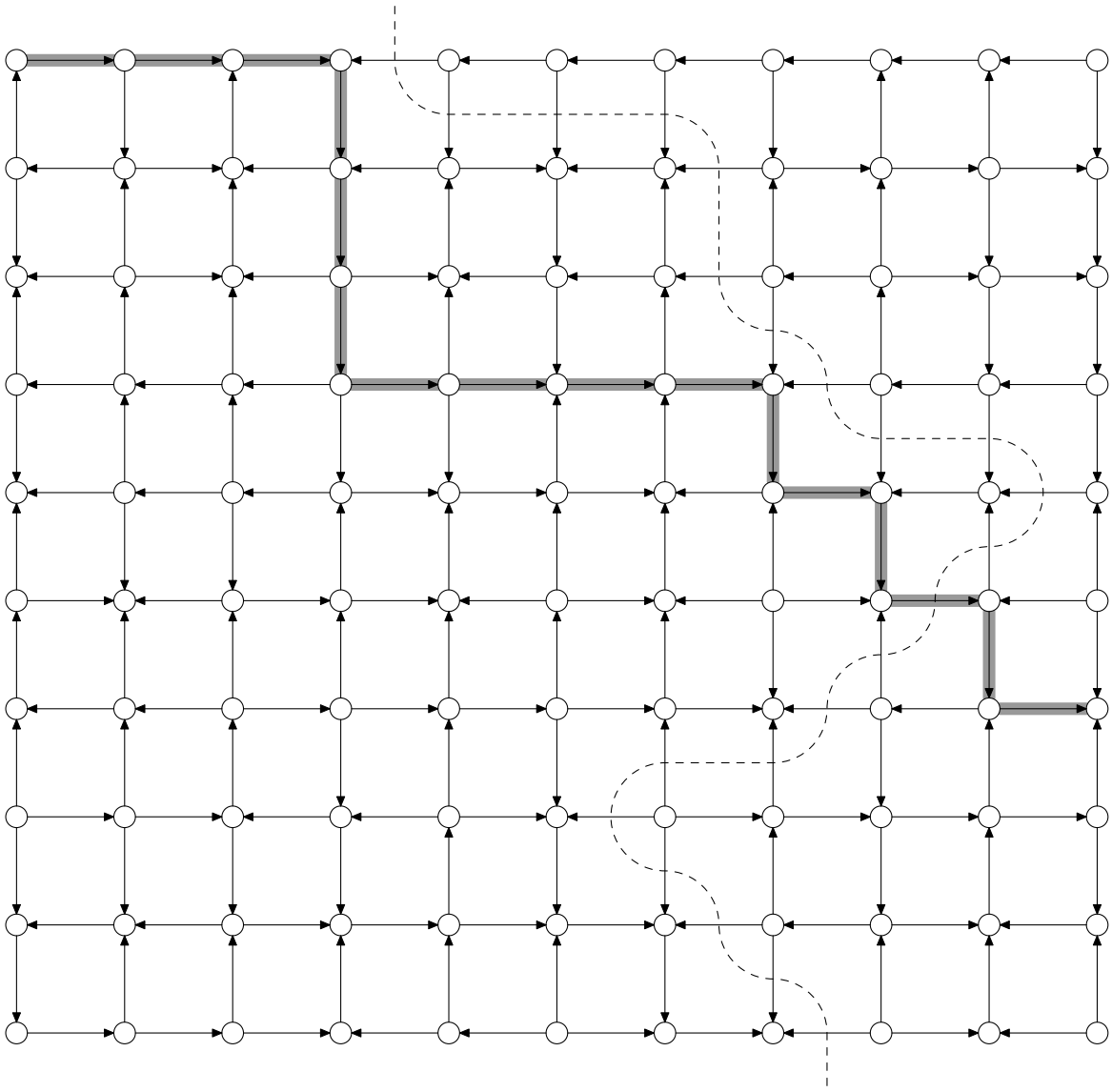


H32, Gruppe A (10 Punkte)

Ermitteln Sie einen maximalen Fluß für das Netzwerk auf der folgenden Seite, wobei die Quellen links und die Senken rechts sind. Beweisen Sie die Maximalität durch Angabe eines gleichgroßen Schnitts. Alle Kapazitäten sind 1.

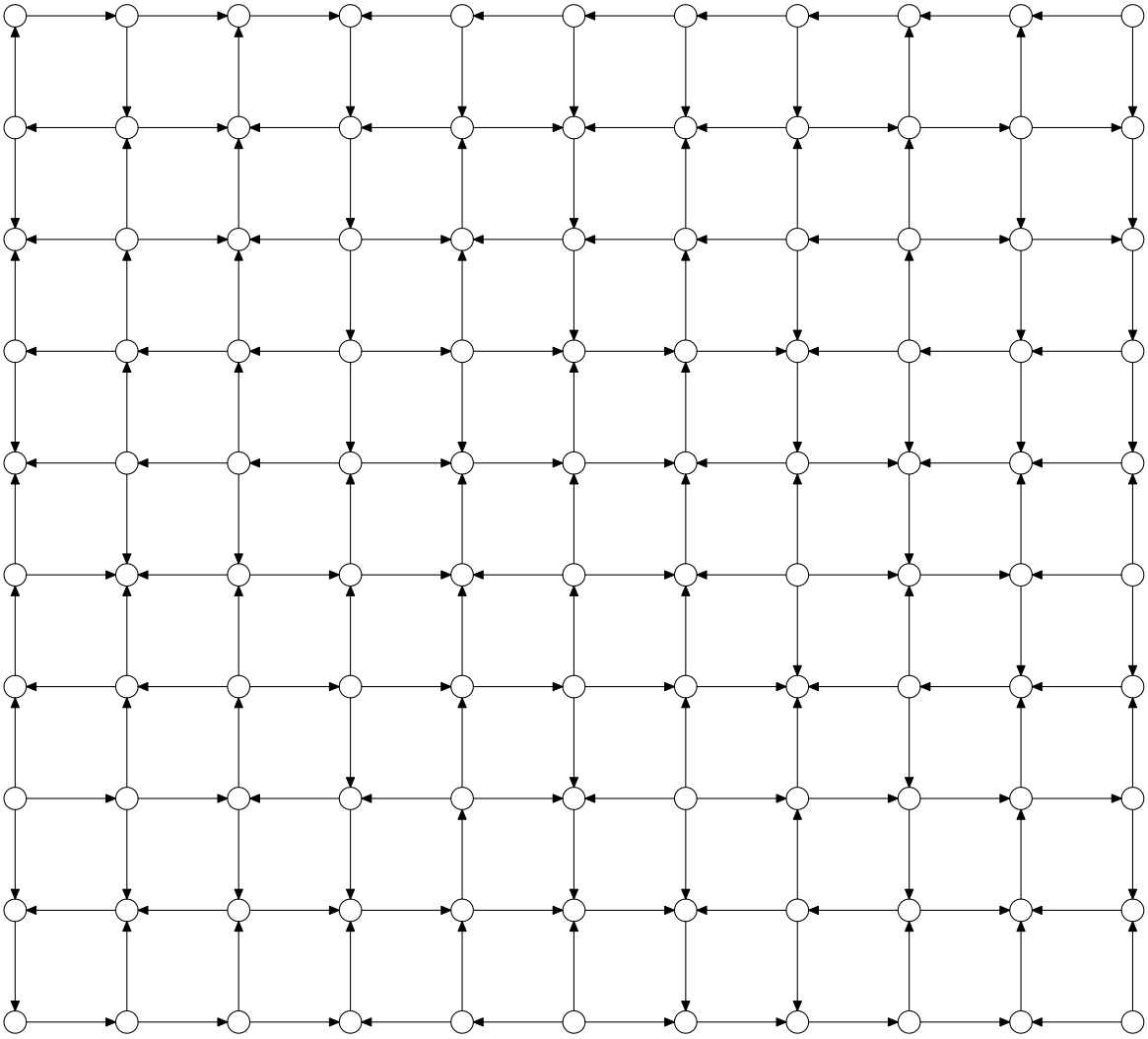


Lösungsvorschlag:

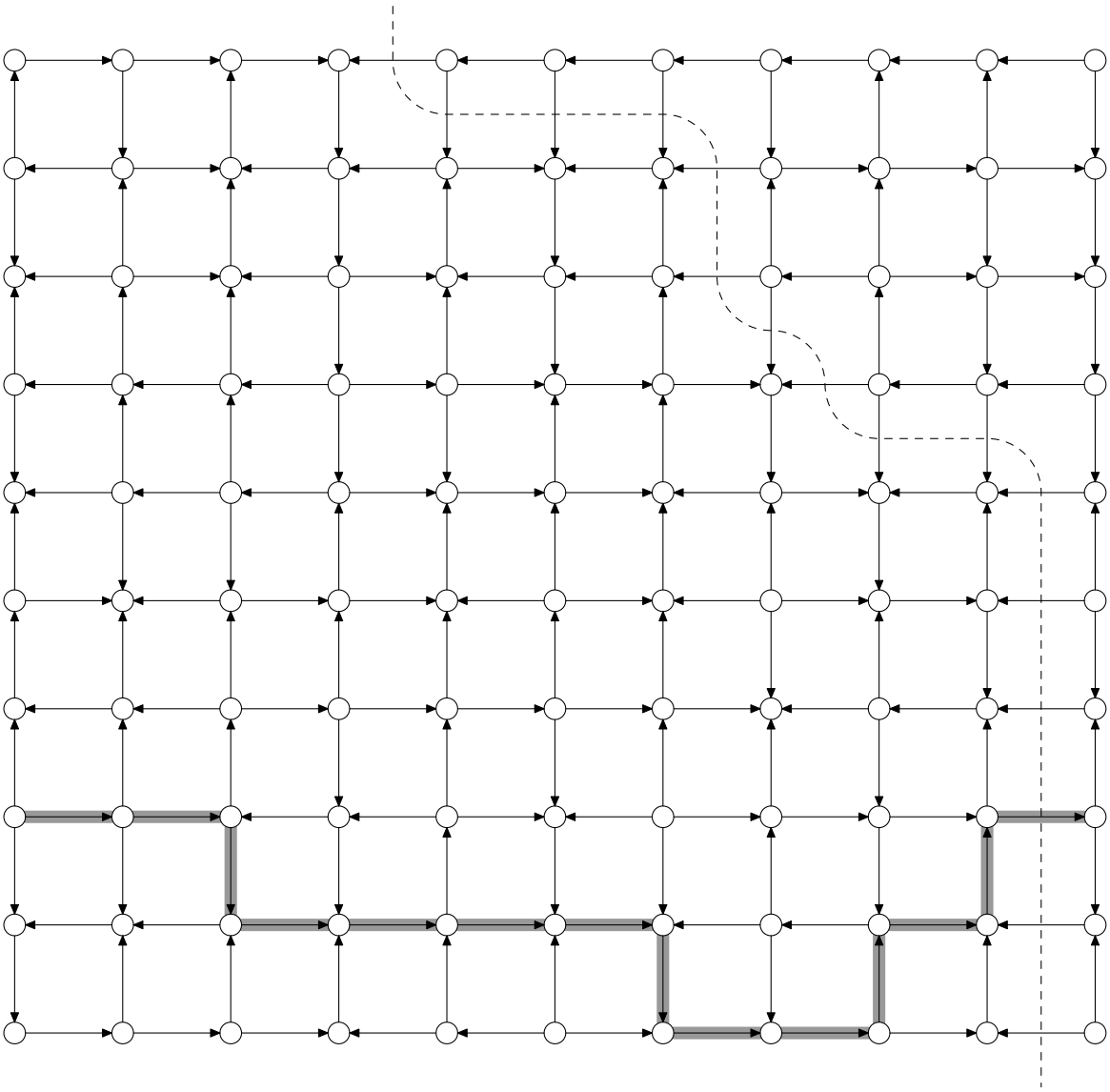


H32, Gruppe B (10 Punkte)

Ermitteln Sie einen maximalen Fluß für das Netzwerk auf der folgenden Seite, wobei die Quellen links und die Senken rechts sind. Beweisen Sie die Maximalität durch Angabe eines gleichgroßen Schnitts. Alle Kapazitäten sind 1.



Lösungsvorschlag:



RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Übungsblatt 13
Emmes—Langer—Reidl—Rossmanith 11.07.2011

Übung zur Vorlesung Datenstrukturen und Algorithmen

T41

Beweisen Sie die Korrektheit des Algorithmus von Prim. Sie können davon ausgehen, daß die Eingabe ein zusammenhängender Graph ist.

Lösung:

Zunächst überprüfen wir, daß die Ausgabe tatsächlich ein Baum ist. Die erste Zwischenlösung, ein einzelner Knoten, ist sicherlich ein zusammenhängend und kreisfrei. In jeder Iteration wird nun ein zusätzlicher Knoten an den schon bestehenden Baum gehängt, dies kann keinen Kreis erzeugen und erhält den Zusammenhang der Lösung (daß ein solcher Knoten existieren muss, folgt aus dem Zusammenhang der Eingabe).

Inbesondere ist also die Ausgabe des Algorithmus ein Baum. Da nur Kanten des Ursprungsgraphen benutzt wurden, ist dieser Baum natürlich ein Spannbaum des Graphen.

Betrachten wir nun eine Ausgabe T des Algorithmus und vergleichen Sie mit einem minimalen Spannbaum T_{min}. Ist $T = T_{min}$ so sind wir fertig. Nehmen wir also an, $T \neq T_{min}$. Es muß dann eine Iteration i gegeben haben, in welcher eine Kante e zu dem aktuellen Spannbaum $T_i \subseteq T$ hinzugefügt wurde, welche nicht in T_{min} enthalten ist (also $T_i \subseteq T_{min}$ gilt jedoch $T_{i+1} = T_i + e \not\subseteq T_{min}$). Bezeichen wir die Knotenmenge $V(T_i)$, auf denen T und T_{min} noch übereinstimmen, mit X.

Betrachten wir die Endpunkte u, v von e. O.b.d.A sei $v \in X$ und $w \notin X$. Es muß in T einen Pfad von u nach v geben, der an irgendeinem Punkt X über eine Kante e' verlässt. Da e' nicht vom Algorithmus gewählt wurde, wissen wir, daß $w(e) \leq w(e')$ gelten muß. Dann aber können wir einen neuen minimalen Spannbaum T'_{min} angeben, der e entält: wir tauschen in T_{min} schlicht e' gegen e aus (man überzeuge sich davon, daß dies tatsächlich einen Baum ergibt!).

Nun können wir induktiv fortfahren: ist $T'_{min} = T$ so sind wir fertig, andernfalls gibt es wiederum eine Iteration $j > i$ in welcher eine Kante zu T hinzugefügt wird, die nicht in T'_{min} enthalten ist, usw.

Dieser Prozess terminiert nach maximal n Schritten und zwar mit einem T''_{min} welcher identisch zu T ist, damit folgt die Minimalität von T.

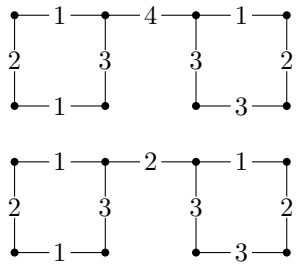
T42

Beweisen Sie Lemma A aus der Vorlesung für 2¢- und 5¢-Münzen: „Sei C eine Münze und v ein Betrag, der mindestens so groß ist wie der Wert von C. Dann ist es suboptimal, den Betrag v mit Münzen kleiner als C auszudrücken.“

Lösung:
2¢-Münzen:
Nehmen wir an, v sei mindestens 2. Da der Betrag durch 1¢-Münzen ausgedrückt wird (nach Annahme wird keine 2¢-Münze benutzt), sind mindestens zwei 1¢-Münzen vorhanden, die wir durch eine 2¢-Münze ersetzen können.
5¢-Münzen:
Nehmen wir nun an, v sei mindestens 5. Werden keine 5¢-Münzen benutzt, so läßt sich der Betrag darstellen als $v = 2c_2 + c_1$, wobei c_2 die Anzahl der 2¢- und c_1 die Anzahl der 1¢-Münzen bezeichnet. Die Anzahl c_1 kann nur die Werte 0 und 1 annehmen, denn ansonsten ist leicht eine kleiner Lösung mit einer weiteren 2¢-Münze gefunden (vgl. die zuvor gezeigte Optimaliztät für 2¢-Münzen). Betrachten wir diese beiden Fälle also separat:

- $c_1 = 0$: Aus $v \geq 5$ folgt, daß $c_2 \geq 3$ ist. Dann aber erhalten wir eine bessere Lösung, wenn wir drei 2¢-Münzen durch eine 5¢- und eine 1¢-Münze ersetzen.
- $c_1 = 1$: Aus $v \geq 5$ folgt, daß $c_2 \geq 2$ ist. Wir erhalten eine bessere Lösung, indem wir zwei 2¢-Münzen und die 1¢-Münze eine 5¢-Münze ersetzen.

H33 (0 Punkte; wird nicht korrigiert)
Sei $G = (V, E)$ ein Graph und $F \subseteq E$ eine Kantenmenge. Wir sagen, F enthält pro Komponente nur einen Kreis genau dann, wenn jede Zusammenhangskomponente von $G = (V, F)$ nur einen Kreis enthält.
Finden Sie ein Verfahren mit Laufzeit $O(m \log n)$, welches für einen gegebenen, kantengewichteten Graphen $G = (V, E)$ mit $n = |V|$ Knoten, $m = |E|$ Kanten und einer Gewichtsfunktion $w\colon E \to \mathbf{N}$ eine Kantenmenge $F \subseteq E$ maximalen Gewichts findet, so daß F pro Komponente nur einen Kreis enthält. Beweisen Sie Korrektheit des Algorithmus und die erreichte Laufzeit.
Sie dürfen verwenden, daß $(E, \{\, F \subseteq E \mid F$ enthält pro Komponente nur einen Kreis $\})$ ein Matroid ist.



Führen Sie Ihren Algorithmus auf den obigen zwei Graphen aus. Wie sieht die optimale Lösung aus?

Lösungsvorschlag:
Wir verwenden einen Greedyalgorithmus, welcher, solange noch nicht betrachtete Kanten existieren, diese entweder zur Menge F hinzufügt oder verwirft. Dieses geschieht in absteigender Reihenfolge nach Gewicht der Kanten. Eine Kante wird F hinzugefügt, falls F keine zwei Komponenten verbindet, die bereits beide Kreise enthalten,und auch keinen zweiten Kreis in einer Komponente erzeugt; sonst wird diese Kante verworfen.
Algorithmus in Pseudo-Code:

1. Sortiere Kanten absteigend nach Gewicht, nenne sie $e_1, \ldots, e_m$.
2. Setze $F := \emptyset$ und $\mathit{kreis}[v] := 0$, $\mathit{comp}[v] := v$ für alle $v \in V$.
3. Für $1 \leq i \leq m$: Sei $e_i = \{u_i, v_i\}$.
   (a) Falls $\mathit{comp}[u_i] = \mathit{comp}[v_i]$ und $\mathit{kreis}[\mathit{comp}[u_i]] = 0$ (gleiche Komponente, noch kein Kreis vorhanden), setze $F := F \cup \{e_i\}$ und $\mathit{kreis}[\mathit{comp}[u_i]] := 1$.
   (b) Falls $\mathit{comp}[u_i] \neq \mathit{comp}[v_i]$ und $\mathit{kreis}[\mathit{comp}[u_i]] \wedge \mathit{kreis}[\mathit{comp}[v_i]] = 0$ (verschiedene Komponenten, wenigstens eine bisher ohne Kreis), setze $F := F \cup \{e_i\}$, $\mathit{comp}[v_i] := \mathit{comp}[u_i]$ sowie

$$\mathit{kreis}[\mathit{comp}[u_i]] := \mathit{kreis}[\mathit{comp}[u_i]] \vee \mathit{kreis}[\mathit{comp}[v_i]] = 0.$$

Die Korrektheit dieses Algorithmus folgt unmittelbar aus der in der Vorlesung bewiesenen Lemmata zu den auf gewichteten Matroiden basierenden Greedy-Algorithmen. Das Sortieren der Kanten benötigt Zeit $O(m \log m) = O(m \log n)$, da $m \leq n^2$ gilt. Die Schleife wird m mal ausgeführt. Die Komponenten eines Knotens können mittels einer Union-Find-Datenstruktur effizient in Zeit $O(\log n)$ ermittelt und aktualisiert werden, so daß die Gesamtlaufzeit tatsächlich $O(m \log n)$ beträgt.

RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Blatt 1
Emmes—Langer—Reidl—Rossmanith 18.4.2011

Übung zur Vorlesung Datenstrukturen und Algorithmen

Eigenständige Präsenzübung (Gruppe A)

Name: _____ Matrikelnummer: _____

Alle Antworten sind zu beweisen!

Aufgabe 1 (10 Punkte)

Gegeben sei ein Array $a[n]$ mit n Elementen $a[0]$ bis $a[n-1]$. Der folgende Algorithmus soll testen, ob ein Element x im Array enthalten ist oder nicht.

function *find3*(**int** x) **boolean** :
$temp := a[n-1]$;
$a[n-1] := x$;
$i := 0$;
while $a[i] \neq x$ **do** $i := i + 1$ **od**;
$a[n-1] := temp$;
return $i < n - 1$

Leider liefert dieser Algorithmus manchmal das falsche Ergebnis. Erklären Sie, wann und warum dieser Fehler auftritt, und wie man ihn beheben kann.

RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Blatt 1
Emmes—Langer—Reidl—Rossmanith 19.4.2011

Übung zur Vorlesung Datenstrukturen und Algorithmen

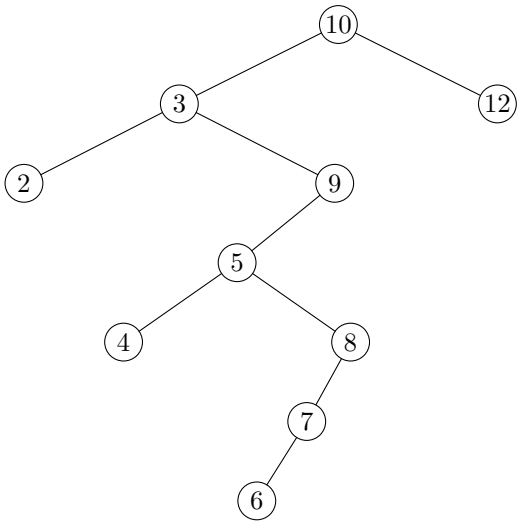
Eigenständige Präsenzübung (Gruppe B)

Name: _____ Matrikelnummer: _____

Alle Antworten sind zu beweisen!

Aufgabe 1 (10 Punkte)

Gegeben sei der folgende binäre Suchbaum:




```
void delete() {
  if(left ≡ null && right ≡ null) {
    if(parent.left ≡ this) parent.left = null;
    else parent.right = null; }
  else if(left ≡ null) {
    if(parent.left ≡ this) parent.left = right;
    else parent.right = right;
    right.parent = parent; }
  else {
    Searchtreenode⟨K, D⟩ max = left;
    while(max.right ≠ null) max = max.right;
    copy(max); max.delete();
  }
}
```

Konstruieren Sie den binären Suchbaum, welcher sich ergibt, wenn das Element 10 gelöscht wird. Als mögliche Hilfe ist die relevante Methode der Klasse *Searchtreenode*$\langle K, D\rangle$ angegeben. Erklären Sie, was hier beim Löschen passiert.

RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Blatt 1
Emmes—Langer—Reidl—Rossmanith 20.4.2011

Übung zur Vorlesung Datenstrukturen und Algorithmen

Eigenständige Präsenzübung (Gruppe C)

Name: _____ Matrikelnummer: _____

Alle Antworten sind zu beweisen!

Aufgabe 1 (10 Punkte)

Gegeben sei ein sortieres Array $a[n]$ mit n Elementen $a[0]$ bis $a[n-1]$, die in aufsteigender Reihenfolge sortiert sind. Betrachten Sie den folgenden Algorithmus, der mit Hilfe binärer Suche testet, ob das Element x im Array enthalten ist.

function *binsearch*(**int** x) **boolean** :
$l := 0; r := n - 1;$
while $l \leq r$ **do**
  $m := \lfloor (l + r)/2 \rfloor;$
  if $a[m] < x$ **then** $l := m + 1$ **fi**;
  if $a[m] > x$ **then** $r := m - 1$ **fi**;
  if $a[m] = x$ **then return** *true* **fi**
od;
return *false*

Beweisen Sie, daß der Algorithmus terminiert. Hinweis: Wie verhält sich $r - l$?

RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Blatt 2
Emmes—Langer—Reidl—Rossmanith 26.4.2011

Übung zur Vorlesung Datenstrukturen und Algorithmen

Eigenständige Präsenzübung (Gruppe A)

Name: _____ Matrikelnummer: _____

Alle Antworten sind zu beweisen!

Aufgabe 1 (10 Punkte)

Die Zahlen 1, 2 und 3 werden in zufälliger Reihenfolge in einen zu Beginn leeren binären Suchbaum eingefügt. Welches ist die erwartete Höhe des entstehenden Suchbaums? Die erwartete Höhe ist die durchschnittliche Höhe der Suchbäume über alle möglichen Einfügereihenfolgen.

RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Blatt 2
Emmes—Langer—Reidl—Rossmanith 27.4.2011

Übung zur Vorlesung Datenstrukturen und Algorithmen

Eigenständige Präsenzübung (Gruppe B)

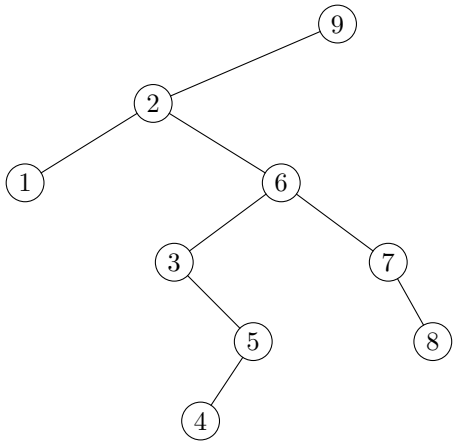
Name: _____ Matrikelnummer: _____

Alle Antworten sind zu beweisen!

Aufgabe 1 (10 Punkte)

Betrachten Sie den binären Suchbaum rechts. In welcher Reihenfolge könnten die Elemente in einen zu Beginn leeren binären Suchbaum eingefügt worden sein, damit dieser Suchbaum entsteht?

Konstruieren Sie sodann den binären Suchbaum, welcher sich ergibt, wenn nun das Element 6 gelöscht wird.



RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Blatt 3
Emmes—Langer—Reidl—Rossmanith 02.05.2011

Übung zur Vorlesung Datenstrukturen und Algorithmen

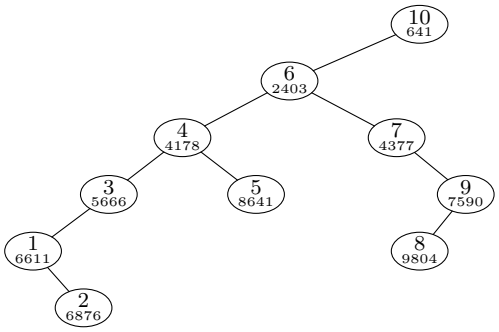
Eigenständige Präsenzübung (Gruppe A)

Name: _____ Matrikelnummer: _____

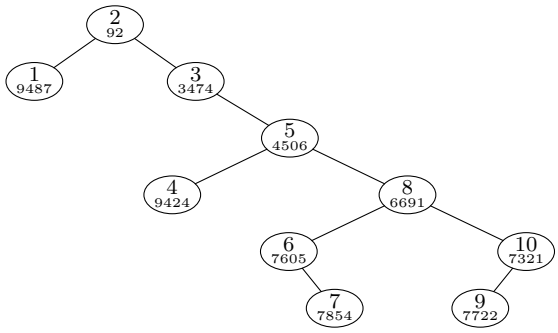
Alle Antworten sind zu beweisen!

Aufgabe 1 (10 Punkte)

Konstruieren Sie den Treap, der sich aus dem unten links abgebildeten Treap ergibt, wenn das Element 6 gelöscht wird.



Konstruieren Sie den Treap, der sich aus dem unten rechts abgebildeten Treap ergibt, wenn das Element 11 mit Priorität 5831 eingefügt wird.



RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Blatt 3
Emmes—Langer—Reidl—Rossmanith 03.05.2011

Übung zur Vorlesung Datenstrukturen und Algorithmen

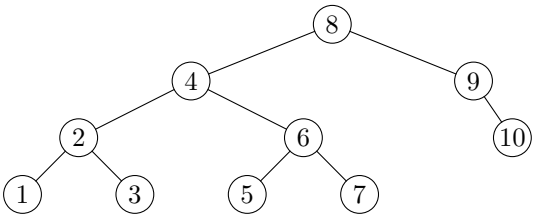
Eigenständige Präsenzübung (Gruppe B)

Name: _____ Matrikelnummer: _____

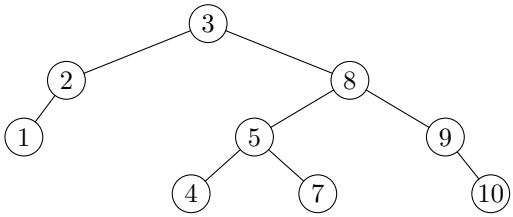
Alle Antworten sind zu beweisen!

Aufgabe 1 (10 Punkte)

Konstruieren Sie den AVL-Baum, der sich aus dem unten links abgebildeten AVL-Baum ergibt, wenn das Element 9 gelöscht wird.



Konstruieren Sie den AVL-Baum, der sich aus dem unten rechts abgebildeten AVL-Baum ergibt, wenn das Element 6 eingefügt wird.



RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Blatt 4
Emmes—Langer—Reidl—Rossmanith 09.05.2011

Übung zur Vorlesung Datenstrukturen und Algorithmen

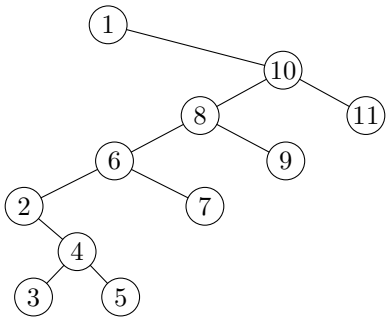
Eigenständige Präsenzübung (Gruppe A)

Name: _____ Matrikelnummer: _____

Alle Antworten sind zu beweisen!

Aufgabe 1 (10 Punkte)

Gegeben sei folgender Splay-Tree:



Es wird der Schlüssel 6 gelöscht. Wie sieht der Splay-Tree anschließend aus?

RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Blatt 4
Emmes—Langer—Reidl—Rossmanith 10.05.2011

Übung zur Vorlesung Datenstrukturen und Algorithmen

Eigenständige Präsenzübung (Gruppe B)

Name: _____ Matrikelnummer: _____

Alle Antworten sind zu beweisen!

Aufgabe 1 (10 Punkte)

Konstruieren Sie einen optimalen Suchbaum, der die Schlüssel 1, 2 und 3 enthält. Die jeweiligen Zugriffswahrscheinlichkeiten mögen 1/4, 3/8 und 3/8 betragen. Erstellen Sie dazu die Tabellen für die $w_{i,j}$ und $e_{i,j}$.

RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Blatt 4
Emmes—Langer—Reidl—Rossmanith 11.05.2011

Übung zur Vorlesung Datenstrukturen und Algorithmen

Eigenständige Präsenzübung (Gruppe C)

Name: _____ Matrikelnummer: _____

Alle Antworten sind zu beweisen!

Aufgabe 1 (10 Punkte)

Konstruieren Sie einen optimalen Suchbaum, der die Schlüssel 1, 2 und 3 enthält. Die jeweiligen Zugriffswahrscheinlichkeiten mögen 1/2, 1/8 und 3/8 betragen. Erstellen Sie dazu die Tabellen für die $w_{i,j}$ und $e_{i,j}$.

RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Blatt 5
Emmes—Langer—Reidl—Rossmanith 16.05.2011

Übung zur Vorlesung Datenstrukturen und Algorithmen

Eigenständige Präsenzübung (Gruppe A)

Name: _____ Matrikelnummer: _____

Alle Antworten sind zu beweisen!

Aufgabe 1 (10 Punkte)

Wenn der Mischalgorithmus der Vorlesung auf folgende zwei Zahlenfolgen angewendet wird, was ist das Resultat?

- 10, 23, 4, 34, 45, 90, 34
- 4, 89, 34, 23, 12, 34, 78

Erklären Sie, wie der Algorithmus vorgeht.
(Innerhalb von Mergesort wird Mischen nur auf sortierte Folgen angewendet, was hier nicht der Fall ist.)

RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Blatt 5
Emmes—Langer—Reidl—Rossmanith 17.05.2011

Übung zur Vorlesung Datenstrukturen und Algorithmen

Eigenständige Präsenzübung (Gruppe B)

Name: _____ Matrikelnummer: _____

Alle Antworten sind zu beweisen!

Aufgabe 1 (10 Punkte)

Wenn der Mischalgorithmus der Vorlesung auf folgende zwei Zahlenfolgen angewendet wird, was ist das Resultat?

- 10, 23, 4, 34, 45, 90, 34
- 4, 89, 34, 23, 12, 34, 78

Erklären Sie, wie der Algorithmus vorgeht.
(Innerhalb von Mergesort wird Mischen nur auf sortierte Folgen angewendet, was hier nicht der Fall ist.)

RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Blatt 5
Emmes—Langer—Reidl—Rossmanith 18.05.2011

Übung zur Vorlesung Datenstrukturen und Algorithmen

Eigenständige Präsenzübung (Gruppe C)

Name: _____ Matrikelnummer: _____

Alle Antworten sind zu beweisen!

Aufgabe 1 (10 Punkte)

Erklären Sie, was eine amortisierte Analyse ist, wozu sie gebraucht wird und wie man sie im einzelnen durchführt.

Welche Vorteile hat sie gebenüber einer klassischen Analyse? Welche Nachteile hat sie?

RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Blatt 6
Emmes—Langer—Reidl—Rossmanith 23.05.2011

Übung zur Vorlesung Datenstrukturen und Algorithmen

Eigenständige Präsenzübung (Gruppe A)

Name: _____ Matrikelnummer: _____

Aufgabe 1 (10 Punkte)

Benutzen Sie Quicksort, um das folgende Array aufsteigend zu sortieren

$$5, 1, 3, 10, 5, 7, 6$$

Verwenden Sie jeweils das Element mit dem kleinsten Index als Pivotelement.
Achten Sie darauf, daß alle Schritte gut nachvollziehbar sind!

RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Blatt 6
Emmes—Langer—Reidl—Rossmanith 24.05.2011

Übung zur Vorlesung Datenstrukturen und Algorithmen

Eigenständige Präsenzübung (Gruppe B)

Name: _____ Matrikelnummer: _____

Aufgabe 1 (10 Punkte)

Benutzen Sie Radix-Exchange-Sort, um das folgende Array aufsteigend zu sortieren

$$5, 1, 3, 4, 7, 6, 2$$

Achten Sie darauf, daß alle Schritte gut nachvollziehbar sind!

RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Blatt 6
Emmes—Langer—Reidl—Rossmanith 25.05.2011

Übung zur Vorlesung Datenstrukturen und Algorithmen

Eigenständige Präsenzübung (Gruppe C)

Name: _____ Matrikelnummer: _____

Aufgabe 1 (10 Punkte)

Benutzen Sie Heapsort, um das folgende Array aufsteigend zu sortieren

$$5, 1, 3, 10, 5, 7, 6$$

Achten Sie darauf, daß alle Schritte gut nachvollziehbar sind!

RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Blatt 7
Emmes—Langer—Reidl—Rossmanith 30.05.2011

Übung zur Vorlesung Datenstrukturen und Algorithmen

Eigenständige Präsenzübung (Gruppe A)

Name: _____ Matrikelnummer: _____

Alle Antworten sind zu beweisen!

Aufgabe 1 (10 Punkte)

```
class Listnode⟨K, D⟩ {
  K key;
  D data;
  Listnode⟨K, D⟩ pred, succ;
  Listnode(K k, D d) {
    key = k; data = d; pred = null;
    succ = null;
  }
  void delete() {
    pred.succ = succ; succ.pred = pred;
  }
  void append(Listnode⟨K, D⟩ newnode) {
    newnode.succ = succ;
    newnode.pred = this;
    succ = newnode;
  }
}
```

In der Vorlesung wurde eine Beispielimplementierung für eine doppelt verkettete Liste vorgestellt. Betrachten Sie den Quellcode links für die Klasse eines zugehörigen Listenknotens, welcher leider einen Fehler enthält: Wenn n ein bereits in der verketteten Liste enthaltener Listenknoten ist, dann sollte ein Aufruf $n.append(\textbf{new } Listnode\langle K, D\rangle(key, value))$ eigentlich einen neuen Listenknoten mit den gegebenen Werten *key* und *value* hinter den aktuellen Knoten n in die Liste einfügen, wie es das Bild unten veranschaulicht. Leider ist die Implementation fehlerhaft, so daß der Aufruf von *append* nicht das gewünschte tut. Erklären Sie, wo der Fehler liegt, und wie man ihn beheben kann.

Liste vorher:

$$\cdots \rightleftarrows (k_1, v_1) \rightleftarrows (k_2, v_2) \rightleftarrows (k_3, v_3) \rightleftarrows \overbrace{(k_4, v_4)}^{n} \rightleftarrows (k_5, v_5) \rightleftarrows \cdots$$

Gewünschtes Ergebnis:

$$\cdots \rightleftarrows (k_1, v_1) \rightleftarrows (k_2, v_2) \rightleftarrows (k_3, v_3) \rightleftarrows \overbrace{(k_4, v_4)}^{n} \rightleftarrows \overbrace{(key, value)}^{\text{neues Element}} \rightleftarrows (k_5, v_5) \rightleftarrows \cdots$$

RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Blatt 7
Emmes—Langer—Reidl—Rossmanith 31.05.2011

Übung zur Vorlesung Datenstrukturen und Algorithmen

Eigenständige Präsenzübung (Gruppe B)

Name: _____ Matrikelnummer: _____

Alle Antworten sind zu beweisen!

Aufgabe 1 (10 Punkte)

Gegeben sei ein Array a mit n Elementen, welches folgende Eigenschaft hat: Es ist derzeit nicht sortiert, läßt sich jedoch durch den *einmaligen* Tausch zweier Elemente i, j in eine sortierte Form überführen. Man muß also nur einmal $swap(i, j)$ auf a aufrufen, um ein sortiertes Array zu erhalten.

Leider wissen Sie nicht, um welche Elemente i, j es sich handelt. Entwerfen Sie einen Algorithmus, der für ein gegebenes Array mit n Elementen, welches obige Eigenschaft hat, in $O(n)$ Schritten feststellt, welche zwei Elemente i, j vertauscht werden müssen, so daß a danach sortiert ist. Sie dürfen Pseudocode oder richtigen Code angeben oder den Algorithmus umgangssprachlich beschreiben. Begründen Sie jedoch ausführlich, warum Ihr Algorithmus die richtigen Elemente in $O(n)$ Schritten findet.

RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Blatt 7
Emmes—Langer—Reidl—Rossmanith 01.06.2011

Übung zur Vorlesung Datenstrukturen und Algorithmen

Eigenständige Präsenzübung (Gruppe C)

Name: _____ Matrikelnummer: _____

Alle Antworten sind zu beweisen!

Aufgabe 1 (10 Punkte)

Beweisen oder widerlegen Sie:

$$n \log n = O\left((\sqrt{n})^{\log n}\right)$$

RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Blatt 8
Emmes—Langer—Reidl—Rossmanith 06.06.2011

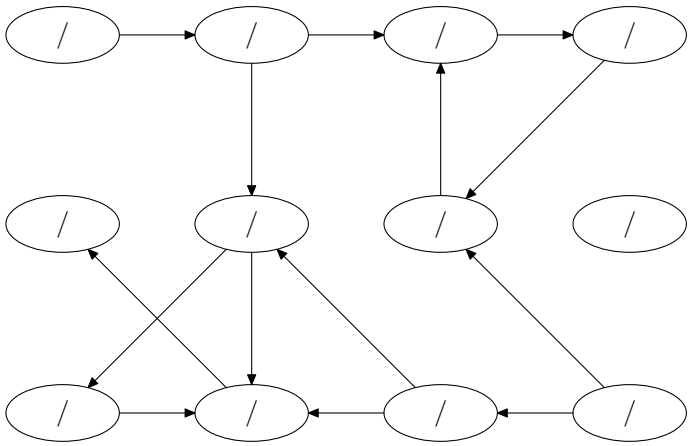
Übung zur Vorlesung Datenstrukturen und Algorithmen

Eigenständige Präsenzübung (Gruppe A)

Name: _____ Matrikelnummer: _____

Aufgabe 1 (10 Punkte)

Führen Sie auf folgendem Graphen eine Tiefensuche durch und markieren Sie dabei jeden Knoten mit seiner Start- und Endzeit.



RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Blatt 8
Emmes—Langer—Reidl—Rossmanith 07.06.2011

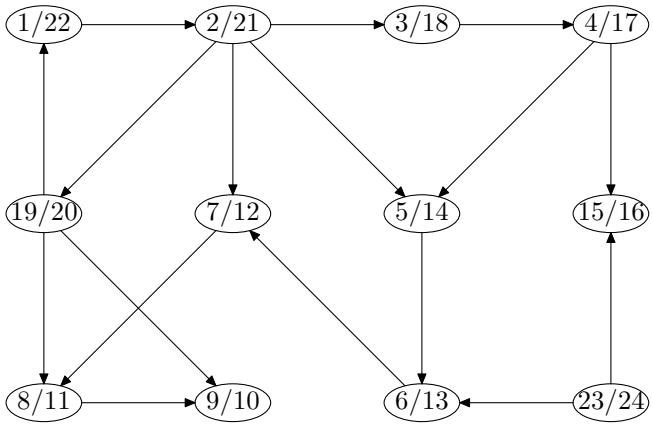
Übung zur Vorlesung Datenstrukturen und Algorithmen

Eigenständige Präsenzübung (Gruppe B)

Name: _____ Matrikelnummer: _____

Aufgabe 1 (10 Punkte)

Der folgende Graph ist mit Start- und Endzeiten einer Tiefensuche beschriftet. Notieren Sie an jeder Kannte, ob es sich um eine **B**aum-, **V**orwärts-, **R**ückwärts- oder **Q**uerkante handelt.



RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Blatt 8
Emmes—Langer—Reidl—Rossmanith 08.06.2011

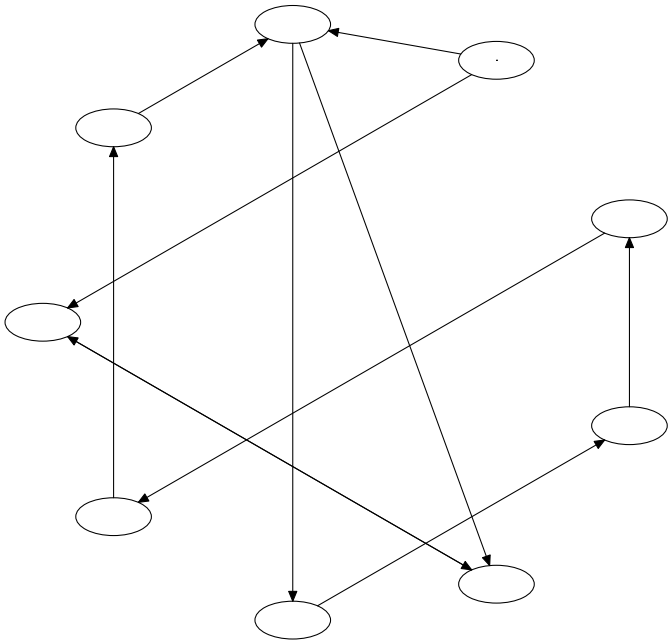
Übung zur Vorlesung Datenstrukturen und Algorithmen

Eigenständige Präsenzübung (Gruppe C)

Name: _____ Matrikelnummer: _____

Aufgabe 1 (10 Punkte)

Bestimmen Sie die starken Zusammenhangskomponenten des folgenden Graphen, indem Sie den Algorithmus von Kosaraju verwenden.



RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Blatt 9
Emmes—Langer—Reidl—Rossmanith 20.06.2011

Übung zur Vorlesung Datenstrukturen und Algorithmen

Eigenständige Präsenzübung (Gruppe A)

Name: _____ Matrikelnummer: _____

Aufgabe 1 (10 Punkte)

Implementieren Sie die Einfügeoperation für binäre Suchbäume in Pseudocode.

RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Blatt 9
Emmes—Langer—Reidl—Rossmanith 21.06.2011

Übung zur Vorlesung Datenstrukturen und Algorithmen

Eigenständige Präsenzübung (Gruppe B)

Name: _____ Matrikelnummer: _____

Aufgabe 1 (10 Punkte)

Implementieren Sie in Pseudocode einen Algorithmus, der auf einem gegebenen Graphen eine Tiefensuche durchführt.

RWTH Aachen
Lehrgebiet Theoretische Informatik
Emmes—Langer—Reidl—Rossmanith

SS 2011
Blatt 9
22.06.2011

Übung zur Vorlesung Datenstrukturen und Algorithmen

Eigenständige Präsenzübung (Gruppe C)

Name: _____ Matrikelnummer: _____

Aufgabe 1 (10 Punkte)

Implementieren Sie in Pseudocode einen Algorithmus, der auf einem gegebenen sortierten Array eine binäre Suche nach einem Element durchführt und dabei ohne rekursive Aufrufe auskommt.

RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Blatt 10
Emmes—Langer—Reidl—Rossmanith 27.06.2011

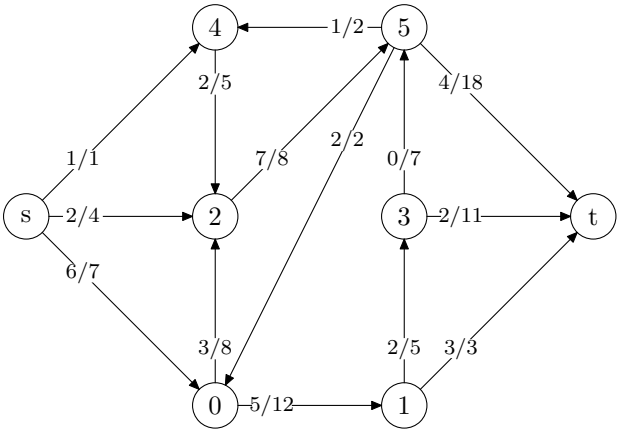
Übung zur Vorlesung Datenstrukturen und Algorithmen

Eigenständige Präsenzübung (Gruppe A)

Name: _____ Matrikelnummer: _____

Aufgabe 1 (10 Punkte)

Bestimmen Sie das Residualnetzwerk des folgenden Netzwerkes mit angegebenem Fluß f.



RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Blatt 10
Emmes—Langer—Reidl—Rossmanith 28.06.2011

Übung zur Vorlesung Datenstrukturen und Algorithmen

Eigenständige Präsenzübung (Gruppe B)

Name: _____ Matrikelnummer: _____

Aufgabe 1 (10 Punkte)

Sei f ein Fluß für das s-t-Netzwerk $G = (V, E)$. Beweisen Sie:

$$f(X, X) = 0 \text{ für } X \subseteq V$$

RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Blatt 10
Emmes—Langer—Reidl—Rossmanith 29.06.2011

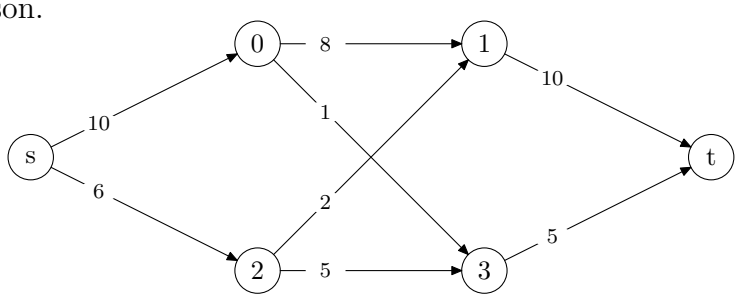
Übung zur Vorlesung Datenstrukturen und Algorithmen

Eigenständige Präsenzübung (Gruppe C)

Name: _____ Matrikelnummer: _____

Aufgabe 1 (10 Punkte)

Bestimmen Sie den maximalen Fluß des folgenden Netzwerkes mit Hilfe der Methode von Ford und Fulkerson.



RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Blatt 11
Emmes—Langer—Reidl—Rossmanith 04.07.2011

Übung zur Vorlesung Datenstrukturen und Algorithmen

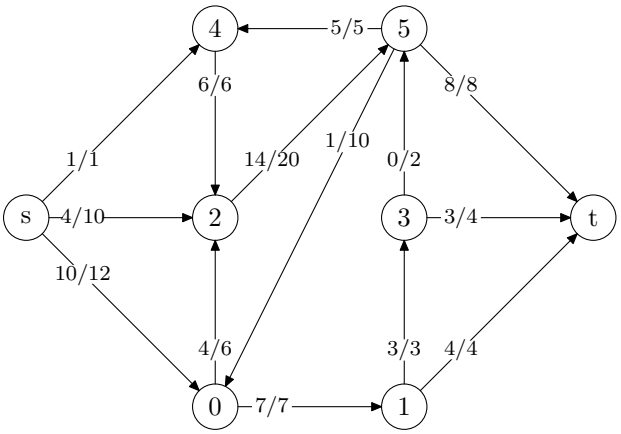
Eigenständige Präsenzübung (Gruppe A)

Name: _____ Matrikelnummer: _____

Aufgabe 1 (10 Punkte)

Erklären Sie, was ein Schnitt in einem Netzerk ist. Gehen Sie auch auf die Defintion minimaler Schnitte ein.

Bestimmen Sie einen minimalen Schnitt im folgenden Netzwerk mit gegebenem Fluß:



RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Blatt 11
Emmes—Langer—Reidl—Rossmanith 05.07.2011

Übung zur Vorlesung Datenstrukturen und Algorithmen

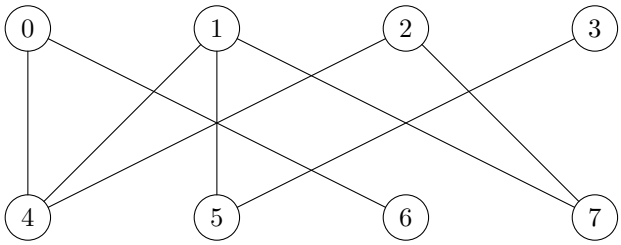
Eigenständige Präsenzübung (Gruppe B)

Name: _____ Matrikelnummer: _____

Aufgabe 1 (10 Punkte)

Erklären Sie, was ein Matching in einem bipartiten Graphen ist und wie man ein maximales Matching in einem solchen Graphen effizient bestimmen kann.

Bestimmen Sie eine maximales Matching in folgendem Graphen:



RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Blatt 11
Emmes—Langer—Reidl—Rossmanith 06.07.2011

Übung zur Vorlesung Datenstrukturen und Algorithmen

Eigenständige Präsenzübung (Gruppe C)

Name: _____ Matrikelnummer: _____

Aufgabe 1 (10 Punkte)

Beweisen Sie folgende Aussage: Wenn ein Fluß in einem Netzwerk maximal ist, dann existiert kein augmentierender Pfad.

RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Blatt 12
Emmes—Langer—Reidl—Rossmanith 11.07.2011

Übung zur Vorlesung Datenstrukturen und Algorithmen

Eigenständige Präsenzübung (Gruppe A)

Name: _____ Matrikelnummer: _____

Beantworten Sie die folgenden Aufgabenstellungen knapp und ohne Beweis.

Aufgabe 1 (10 Punkte)

Hat Quicksort eine Laufzeit von $O(n^2)$?

Wie lange dauert die Suche nach einem Element in einer verketteten Liste der Länge n?

Lösen Sie die folgende Rekursionsgleichung: $a_0 = 2$, $a_n = 3a_{n-1}$ für $n > 0$

Geben Sie ein Beispiel für eine Datenstruktur an, die amortisiert schnell ist, für einzelne Operationen allerdings langsam.

Wieviele Rotationen können beim Einfügen eines Elementes in einen AVL-Baum der Höhe n maximal auftreten?

Was bedeutet es, wenn ein Sortierverfahren *stabil* ist?

Nennen Sie eine Datenstruktur, die sich als *priority queue* eignet.

Wie schnell lässt sich der kürzeste Weg zwischen zwei Knoten in einem Graphen $G = (V, E)$ finden?

Geben Sie ein Flussnetzwerk an, dessen minimaler Schnitt eine Kapazität von 42 hat.

In einen zu Beginn leeren binären Suchbaum werden nacheinander die vier Elemente 1, 4, 3, 2 eingefügt. Wie sieht er nun aus?

RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Blatt 12
Emmes—Langer—Reidl—Rossmanith 12.07.2011

Übung zur Vorlesung Datenstrukturen und Algorithmen

Eigenständige Präsenzübung (Gruppe B)

Name: _____ Matrikelnummer: _____

Beantworten Sie die folgenden Aufgabenstellungen knapp und ohne Beweis.

Aufgabe 1 (10 Punkte)
Hat Quicksort eine Laufzeit von $O(n \log n)$?

Wie lange dauert die binäre Suche nach einem Element in einem sortieren Array mit n Elementen?

Welche *worst-case* Laufzeit hat das Löschen eines Elementes aus einem binären Suchbaum?

Wie lange dauert es, alle Elemente einer Skiplist in sortierter Reihenfolge auszugeben?

Welche Laufzeit hat Mergesort?

Nennen Sie ein Sortierverfahren, welches nicht vergleichsbasiert ist.

In welche vier Kategorien werden die Kanten bei einer Tiefensuche eingeteilt?

Welche gerichteten Graphen lassen sich topologisch sortieren?

Welche Laufzeit hat der Algorithmus von Warshall für einen Graphen $G = (V, E)$?

In einen zu Beginn leeren AVL-Baum werden nacheinander die Elemente 1, 2, 3, 4 eingefügt. Wie sieht er nun aus?

RWTH Aachen **SS 2011**
Lehrgebiet Theoretische Informatik Blatt 12
Emmes—Langer—Reidl—Rossmanith 13.07.2011

Übung zur Vorlesung Datenstrukturen und Algorithmen

Eigenständige Präsenzübung (Gruppe C)

Name: _____ Matrikelnummer: _____

Beantworten Sie die folgenden Aufgabenstellungen knapp und ohne Beweis.

Aufgabe 1 (10 Punkte)

Hat Heapsort eine Laufzeit von $O(n \log n)$?

Wie ist die erwartete Höhe eines (anfangs leeren) Binärbaums, nachdem man n verschiedene, zufällige Elemente eingefügt hat?

Geben Sie eine möglichst knappe obere Schranke für die Höhe eines AVL-Baumes mit n Elementen an.

Ist die Struktur eines Treaps eindeutig, wenn alle Schlüssel und alle Prioritäten verschieden sind?

Welche Laufzeit hat Insertionsort?

Welche der folgenden Sortierverfahren sind stabil? Mergesort, Heapsort, Quicksort.

Wie schnell kann man den Median von n Zahlen berechnen?

Ein Graph hat n Knoten und m Kanten. Wie schnell lassen sich seine Zusammenhangskomponenten finden?

Welche Laufzeit hat der Algorithmus von Bellman und Ford für einen Graphen $G = (V, E)$?

In einen zu Beginn leeren Splay-Baum werden nacheinander die Elemente 1, 2, 3, 4 eingefügt. Wie sieht er nun aus?